AD-A080 055

AD-A080 055

TECHNICAL
LIBRARY

Interactive Electronic Circuit Simulation on Small Computer Systems

by Brian L. Biehl

U.S. Army Electronics Research
and Development Command
Harry Diamond Laboratories
Adelphi, MD 20783

Approved for public release; distribution unlimited.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturers' or trade names does not constitute an official indorsement or approval of the use thereof.

Destroy this repcrt when it is no longer needed. Do not return it to the originator.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER HDL-TM-79-30	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Interactive Electronic Circuit Simulation on Small Computer Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Memorandum
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Brian L. Biehl		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harry Diamond Laboratories 2800 Powder Mill Road Adelphi, MD 20783		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DA: {1T161101A91A 1L162120AH2501
11. CONTROLLING OFFICE NAME AND ADDRESS Commander U.S. Army Materiel Development and Readiness Command Alexandria, VA 22333		12. REPORT DATE November 1979 13. NUMBER OF PAGES 122
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

HDL Project: {A10796, A77895

DRCMS: {61101.91A0011.A1-A1, 612120.H250011

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Circuit simulation
Computer-aided design
Interactive
Minicomputers
Desktop calculators

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The feasibility is determined of using small computer systems (including programmable desktop calculators and minicomputers) for interactive electronic circuit simulation. Several aspects of the simulator architecture are considered: the computer language, the data word format, the computing speed, and the computer memory configuration.

Interactive circuit simulation on programmable desktop calculators is investigated using simulator program BIAS-D (BASIC), written in BASIC for an HP9830A desktop calculator. Analysis techniques are presented which conserve memory and take advantage of the idiosyncrasies of these small computers.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

Interactive-mode circuit simulation and batch-mode circuit simulation on minicomputers are compared relative to the simulator architecture and required simulation speed. The more significant speed- and memory-dependent algorithms used in circuit simulators are compared in detail. Also compared are the execution speeds of several different minicomputer systems, including the HP2100, the PDP 11/45, and the PRIME 400. The speed and memory requirements of these minicomputers executing BIAS-D are compared to an IBM 370/168 also executing BIAS-D.

A new method for computing small-signal frequency response is introduced. Because complex arithmetic is not required, this technique is particularly suited to minicomputer simulators and requires minimal additional memory when implemented in a circuit simulator with a transient analysis capability. The frequency response of both linear and nonlinear circuits can be modeled, as can that of high-Q circuits. Magnitude and phase errors of less than 1 percent and 0.5 degrees, respectively, are easily attainable. Speed ratios between this technique and a conventional ac analysis vary depending on the circuit Q. For circuits with Q less than 1, this ratio is typically 10:1.

CONTENTS

CONTENTS (Cont'd)

APPENDICES

FIGURES

FIGURES (Cont'd)

FIGURES (Cont'd)

TABLES

TABLES (Cont'd)

1. INTRODUCTION

Electronic circuit simulation has traditionally been done by batch or semi-interactive batch methods using large host computers. It has been only within the last two or three years that serious consideration has been given to the possibility of doing circuit analysis with small computer systems such as desktop calculators and minicomputers. Programs such as BIAS-D [1],* MINI-MSINC [2], and BIASL.25 [3] have been written expressly for these small computing systems. Although these analysis programs do not have the analytical capabilities or speed of simulator programs such as SLIC [4], ASTAP [5], or SPICE [6,7] used on large computer systems, they do represent a potentially valuable design aid for the simulation of small circuits (30 to 50 nodes) [8].

The primary intent of this report is to show the practicality of using small computer systems for interactive circuit simulation and to determine the trade-offs which are necessary to achieve a general-purpose (ac, dc, and transient) electronic circuit simulator within the limitations of these small computer systems.

This report shows that interactive circuit simulation is possible on minicomputer systems. On a dedicated small computer system, the major cost of interactive circuit simulation is the engineer's or designer's time. In contrast, in a large computer system at least an equal cost is contributed by computer costs. The "interactive" simulation is emphasized here (as opposed to batch simulation) since this is the most effective way of completing a computer-aided engineering design cycle. Comparisons of the interactive versus batch simulation procedures are included in section 4, where the simulator architecture and the simulator speed are compared.

One might think that a reasonable initial approach to developing a circuit simulator for these small computer systems would be to convert a program such as SPICE into a minicomputer-compatible language. There are, however, many barriers which make this approach both difficult and uneconomical [9]. Differences in the architectures of large computers such as the IBM's and CDC's and those of minicomputers such as the HP's, PDP's, and PRIME's, as well as computational speed differences, are the primary contributors to these programming difficulties. Section 3 includes a brief description of computer architecture as related to circuit simulation.

Most of the work described has evolved in three phases. The initial work, described in section 2, concentrated on a desktop-calculator simulator, BIAS-D [1], using a first-generation BASIC language desktop calculator, the HP9830A. The 16-kbyte available memory posed a severe limitation (15 nodes, 150 elements) on the size of the circuit which could be simulated. Although these efforts were successful in showing that circuit simulation on programmable calculators is possible, they also determined that the speed and memory limitations of these early calculators are too restrictive for a successful interactive simulator.

The second phase, described elsewhere [10], involved converting BIAS-D from BASIC into a minicomputer-compatible FORTRAN IV. Early results on a PRIME 400 minicomputer attained a surprising 600:1 speed improvement over the HP9830A calculator. Significantly more memory was also available, permitting analysis of 30- to 50-node circuits at reasonable speed. This FORTRAN version was used to compare central processor unit (CPU) speeds of several minicomputer systems: the HP2100, the PDP 11/45, and the PRIME 400, as well as the IBM 370/168 [10]. This version of BIAS-D was essentially a conversion of the original BASIC version. No attempt was made to incorporate speed- or memory-saving techniques such as sparse matrix storage and decomposition.

The third and final phase, described in sections 4 and 5, brought together, in BIAS-D, the more significant speed- and memory-saving techniques used in the large computer system circuit-simulator programs: node ordering, sparse matrix decomposition, sparse matrix storage, and linked-

Numbers in brackets refer to the Literature Cited.

list element storage. Detailed comparisons of speed and memory requirements are made for each of these techniques (sect. 4.2 and 4.3). The interactive capabilities of BIAS-D have been enhanced, and small-signal frequency response has been added; this was previously unavailable in a general-purpose minicomputer simulator. Algorithms used in BIAS-3 [11], SLIC [4], SINC [12], and SPICE [6], both published and unpublished, were examined during these efforts.

Section 5 introduces a method for computing ac frequency response which requires no complex arithmetic and very little additional memory. This method uses an extension of the standard transient analysis procedures used in time-domain simulations. Both this new method and the traditional method are implemented in BIAS-D for comparison purposes. Comparisons are made of analytical speed, memory requirements, and accuracy between this new method and the traditional complex-matrix method.

The appendices include user's manuals and source listings for both the BASIC version of BIAS-D for an HP9830A (HP9845) desktop calculator and the FORTRAN version of BIAS-D. The FORTRAN version includes ac analysis and will run on almost any computer system with few if any modifications. Also included are the four benchmark test circuits which were used in many of the timed experiments. A description of the linked-list structure used in BIAS-D to store circuit elements is given, as well as a description of the function of each subroutine in the FORTRAN version of BIAS-D.

2. CIRCUIT SIMULATION ON PROGRAMMABLE CALCULATORS

Even though the programmable desktop calculator does not have the speed or memory capabilities of minicomputers or large computer systems, it is still a convenient, low-cost tool for circuit simulation.

2.1 Circuit Simulation—General

Probably the first general-purpose circuit simulator for programmable calculators was BIAS-D [1]. BIAS-D, written in BASIC for an HP9830A with a 16-kbyte memory, can compute the dc operating points, small-signal ac gain and input impedance, and transient response of a circuit of up to 15 nodes containing resistors, capacitors, current sources, voltage sources, and npn or pnp bipolar transistors (15 each). For transistor circuits, BIAS-D converges to a solution by linearizing the built-in Ebers-Moll transistor model in much the same manner as that done in the larger circuit-simulator programs such as BIAS-3 [11] and SPICE [6].

Subsequently, BIASL.25 [3] was developed for an HP9825 calculator with 32 kbytes of memory. BIASL.25 was developed primarily for simulating metal-oxide-semiconductor (MOS) circuits and consequently has an advanced built-in MOS model. Diodes, resistors, capacitors (both linear and nonlinear), voltage sources, and current sources are also available. The maximum circuit configuration of BIASL.25 is not fixed, because dynamic element allocation is used. A typical circuit would have 19 equations, 14 MOS devices, 5 diodes, 12 capacitors, 5 resistors, and 10 independent sources. The key to BIASL.25's capabilities is a very fast magnetic-tape cassette on the HP9825 which permits extensive program overlays. This cassette allows a much larger program within the limited 32-kbyte real memory. The increased speed of the HP9825 also makes real-time interaction in BIASL.25 more practical than in BIAS-D on the HP9830A. Another feature of these calculators which made both BIAS-D and BIASL.25 practical was the use of built-in read-only memories (ROM's), especially the matrix inversion ROM. The added speed and memory saving of these ROM's makes their use in calculators attractive for circuit simulation. However, incorporation of the features of these ROM's into the circuit-simulator programs can greatly alter the architecture of the program. The use of techniques such as LU decomposition based on matrix sparsity or sparse storage is no longer practical. Some of the alternative techniques

which can be used are presented in section 2.2. The analytical procedures used in the BASIC version of BIAS-D are included here since this was the basis of the ensuing FORTRAN version.

2.2 Circuit Simulation Using BIAS-D (BASIC Version)

Any circuit-simulation program can be divided into three major segments: an input processor, for interpreting the input circuit topology and error checking; a circuit processor, for reconfiguring the circuit for optimum performance in the simulator; and the analysis portion, which solves the circuit equations for each type of analysis.

BIAS-D is written in BASIC for an HP9830A desktop calculator with a 16-kbyte memory, an 80-column printer, a matrix-operations ROM, and a string-variable ROM. However, any calculator system which contains the BASIC interpretive language could be used. In fact, BIAS-D was run on a Tektronix 4051 calculator* and an HP9845 with only minor program changes.

2.2.1 Input Processing

The input language of BIAS-D has been structured to be easy to use and as interactive as possible, and yet use a minimum amount of memory. Whenever possible, the input format has been modeled after that of SPICE [6]. The circuit data are entered into BIAS-D in a semifree format—semifree in that the data must begin in the first column, and a single space must be used as a delimiter between all data fields. Since memory is at a premium, the data images are not stored and, therefore, must be processed as they are entered. This processing must be kept to a minimum to prevent an excessive wait time between data entries. An input flow diagram of this processing is shown in figure 1. As can be seen from this figure, the data path through this routine is determined by the information in one of the first three columns of each data entry. If a permitted character other than a dot (.) appears in the first column, then the ensuing data are those of a circuit

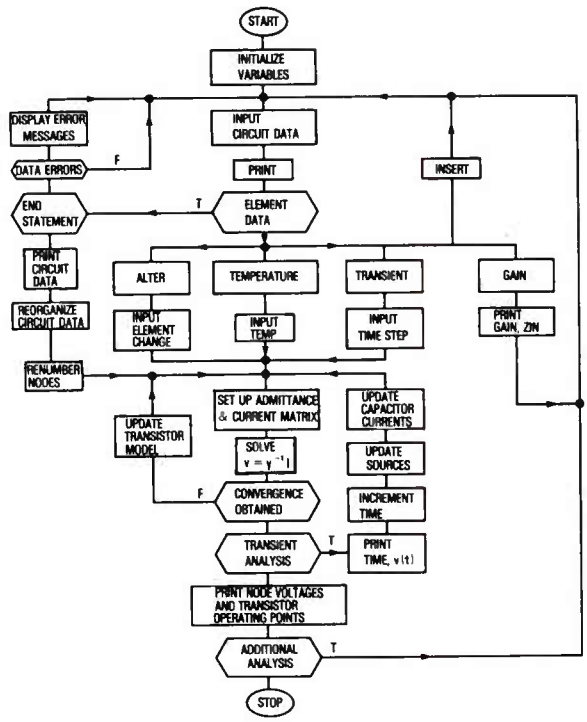


Figure 1. Flow diagram of input processing for BIAS-D.

element (resistor, capacitor, independent voltage or current source, or model), an END statement, or a comment statement. If the first column contains a dot and the second column is a permitted character, then the entry is a control statement (Alter, Insert, Temperature, Transient, Gain, or Output). A number from 1 to 8 is assigned to a flag variable, F, depending on which control statement was entered. This flag is used later in the program to determine which analysis is required. Further data are requested by the program, if necessary, as the analysis proceeds. At the end of each analysis, control is returned to the user for further commands. A sample of this input format is shown in figure 2 (p 12).

2.2.2 Circuit Setup

After the circuit data have been entered and the circuit topology reprinted in an ordered format, BIAS-D then restructures the circuit into a form suitable for analysis.

**B. Ross, Tektronix, Inc., private communication.*

```
* PREAMPLIFIER TEST CIRCUIT
R1 6 1 12000
R2 7 3 7500
R3 4 0 680
R4 7 6 9000
R5 8 0 5000
Q1 3 1 2 M2
Q2 3 2 4 M2
Q3 6 5 4 M2
Q4 6 6 5 M2
Q5 7 3 8 M2
V+ 7 0 6.1
VS 9 0 1.0 M1
RS 9 1 1E8
M1 PUL 1 3 1 30 1 1 1
M2 NPN 100 1 5E-15
END

RESISTORS:
NAME     NODES          VALUE
R1       6         1    12000
R2       7         3    7500
R3       4         0    680
R4       7         6    9000
R5       8         0    5000
RS       9         1    100000000

VOLTAGE SOURCES:
NAME    +NODES-    VALUE  MODEL
V+       7     0     6.1  M 0
VS       9     0     1    M 1

TRANSISTORS:
NAME     C     B     E    MODEL
Q1       3     1     2     M 2
Q2       3     2     4     M 2
Q3       6     5     4     M 2
Q4       6     6     5     M 2
Q5       7     3     8     M 2

MODELS:
NAME  TYPE
M1    PUL    1.000    3.000   1.000E+00   3.000E+01   1.000E+00   1.000E+00
M2    NPN  100.000    1.000   5.000E-15   1.000E+12   0.000E+00   0.000E+00

NODES: 9

****END OF INPUT DATA****

ITERATIONS: 10

T= 27   DEG C

NODE VOLTAGES:
V  1              1.8277
V  2              1.2942
V  3              2.5572
V  4              0.6414
V  5              1.2945
V  6              1.8283
V  7              6.1000
V  8              1.9097
V  9              1.0000
TRANSISTOR OPERATING POINTS:
NAME      IB          IC          VBE          VBC       BETA        GM          RPI
Q1    4.594E-08   4.594E-06    0.533    -0.729      100.00   1.777E-04   5.627E+05
Q2    4.640E-06   4.640E-04    0.653    -1.263      100.00   1.795E-02   5.571E+03
Q3    4.699E-06   4.699E-04    0.653    -0.534      100.00   1.818E-02   5.501E+03
Q4    4.652E-08   4.652E-06    0.534     0.000      100.00   1.800E-04   5.556E+05
Q5    3.782E-06   3.782E-04    0.647    -3.543      100.00   1.463E-02   6.836E+03


.GAIN
INPUT NODE
 1
OUTPUT NODE
 8
GAIN(V/V)=-9.733791498
INPUT IMPEDANCE= 76075.0305

.ALTER
RS 10
END
```

Figure 2. BIAS-D input data and results for preamplifier example circuit from an HP9830 calculator.

During the element data entries, a node set, N_1, is generated which contains all unique node numbers in the particular circuit. Since the elements are entered at random, this node set is not ordered, but the length is known and equal to the number of circuit nodes, N (circuit ground, or node 0, is not included). An additional node set N_2, also of length N, is generated, containing the sequence (1, 2, 3, . . . N). N_1 is then ordered into increasing numerical order. N_1 and N_2 will be used to control the node mapping between the original circuit and the newly processed circuit.

The circuit shown in figure 3 will be used as an example to illustrate the further processing. At this point the node sets N_1 and N_2 for the circuit in figure 3a are

$$N_1 = [\,(1)\,2\ 3\ 5\ 6\ (10)\,] \qquad (1)$$

$$N_2 = [\,1\ 2\ 3\ 4\ 5\ 6\,]$$

The circled nodes in set N_1 are the circuit voltage source nodes. The equivalent voltage source nodes in N_2 are 1 and 6. The source nodes in N_2 are then moved toward the end of N_2 by being exchanged with nonsource nodes. N_1 and N_2 are now

$$N_1 = [\,1\ 2\ 3\ 5\ 6\ 10\,] \qquad (2)$$

$$N_2 = [\,5\ 2\ 3\ 4\ 1\ 6\,]$$

Note that the equivalent voltage source nodes in N_2 are now 5 and 6. The circuit element nodes are next renumbered by converting the original node numbers in N_1 to their equivalents in N_2. The results of this conversion on the example circuit are shown in figure 3b.

The circuit is now restructured by converting elements connected to voltage sources into their Norton equivalents. This is not normally done in the larger circuit codes but is necessary here to avoid manipulating the nodal admittance matrix after it is loaded. For resistors this is accomplished by grounding the node of each resistor connected to these sources and adding a Norton equivalent current source from ground to the other resistor node;

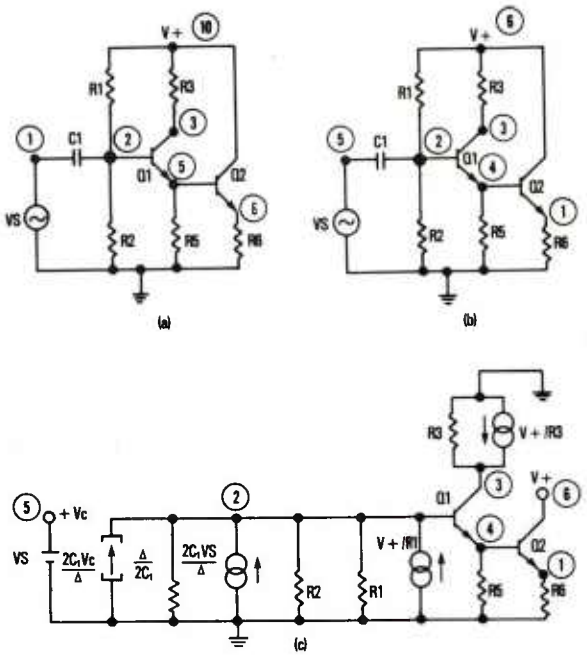


Figure 3. Example circuit showing steps in setup procedure.

current source nodes connected to voltage sources are grounded. A capacitor is represented as a conductance in parallel with a voltage-dependent current source, and capacitors connected to voltage sources are reconfigured by grounding the current source and treating the conductance as a resistor element. Transistors are not altered at this time. Using these conventions, the example circuit shown in figure 3b can be restructured into that of figure 3c. Three additional current sources are added, two for resistors R_1 and R_3 and one for capacitor C_1. The values of these added current sources have been stored symbolically, in the form of either a node number or element value address location, since either the resistor, capacitor, or voltage source values may be altered in subsequent analyses. The circuit is now in its final restructured form (fig. 3c). The known nodes, those of the voltage sources, have been eliminated from the circuit. In the larger circuit-simulator codes, this elimination is done after the admittance matrix has been loaded, and it requires partitioning of the admittance matrix [6]. This is not possible in the available matrix-operations ROM.

2.2.3 Analytical Procedures

The primary analytical procedures involved in circuit simulation are the loading and solving of the matrix equation

$$Y \cdot V = I \quad . \tag{3}$$

The equivalent linear or nonlinear element conductances must be determined and loaded into the nodal admittance matrix, Y; the excitation currents must be determined and loaded into the current vector, I; and equation (3) must be solved for the node voltages, V. These voltages are then used to update Y and I; the procedure is repeated until the process has converged. This procedure requires the most analysis time and memory use in a circuit-simulation program.

Models.—In order to load the nodal admittance matrix in equation (3), the proper model parameters must be determined. These parameters can be a function of time, temperature, or circuit node voltages. In BIAS-D there are five allowable models: two transient source models, a temperature model, and two bipolar transistor models (npn and pnp). For the transistor model, a Newton-Raphson iterative technique is used to determine the parameters. Each model contains six definable parameters plus one which indicates its type. The models are designated by a three-character name as part of a model entry as follows:

MX YYY F1 F2 F3 F4 F5 F6 ,

where M designates that this is a model with a name X and type YYY. F1 through F6 are the model parameters. More details on these parameters and the transient and temperature models are in the BIAS-D (BASIC) user's manual in appendix A.

A modified Ebers-Moll [13] transistor model is used in BIAS-D. A circuit representation of this model (npn) is shown in figure 4a. The large-signal terminal currents are given by

$$\begin{aligned} I_E &= -I_S(1 + 1/B_F)[\exp(V_{BE}/V_T) - 1] \\ &\quad + I_S[\exp(V_{BC}/V_T) - 1] \\ &\quad + I_{RS}[\exp(V_{BE}/2V_T) - 1] \quad , \end{aligned} \tag{4}$$

$$\begin{aligned} I_C &= I_S[\exp(V_{BE}/V_T) - 1] \\ &\quad + I_S(1 + 1/B_R)[\exp(V_{BC}/V_T) - 1] \quad , \end{aligned} \tag{5}$$

$$I_B = I_E - I_C \quad , \tag{6}$$

where

B_F, B_R = forward and reverse dc beta, respectively,
I_S = short circuit saturation current,
I_{RS} = recombination saturation current, and
V_T = kT/q with Boltzmann constant, k, temperature, T, and electronic charge, q.

The last term in equation (4) accounts for the current dependence of beta at low currents. The lower collector knee current, I_L, at which B_F is half of its maximum value (assuming high-level injection effects are negligible in this current range) is [14]

$$I_L = B_{F2} I_{RS2} / I_S \quad . \tag{7}$$

In order to include base-width modulation effects, the saturation current and beta's are multiplied by the term

$$(1 + V_{CB}/V_A); \qquad V_A > V_{CB} \tag{8}$$

where V_A is the early voltage [15]. During the analysis procedure, this large-signal model is linearized about the dc operating points determined from the last computed node voltages. This linearized equivalent model is shown in figure 4b. The transconductance, g_{mx}, and input conductances, $g_{\pi x}$, are obtained by evaluating the appropriate derivatives of equations (4) and (5) at the operating points. The nonlinear junction capacitances are not included in this model, but could be included as separate linear capacitors with a value determined by the junction voltages computed at the circuit operation points.

Matrix setup and inversion.—In addition to the circuit restructuring, which eliminates the voltage source nodes as was described earlier, another method of saving computation time was found. In this method, the nodal admittance matrix is loaded as a definite admittance matrix rather than an indefinite admittance matrix, which is normally loaded in a non-sparse-solution method. The memory re-

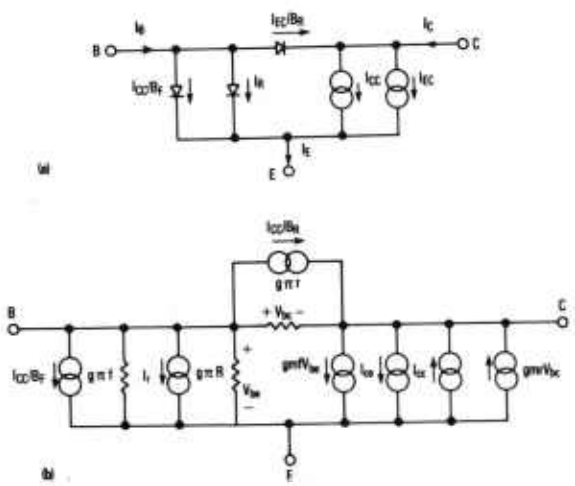


Figure 4. Ebers-Moll transistor model: (a) large-signal model and (b) linearized model.

quirements for both methods were approximately the same since the additional coding required to implement the definite matrix offset the 2N memory saving because of the elimination of a node. For the circuit shown in figure 5 the setup and inversion time for the determinate matrix form was about 25 percent less than that for the indeterminate matrix.

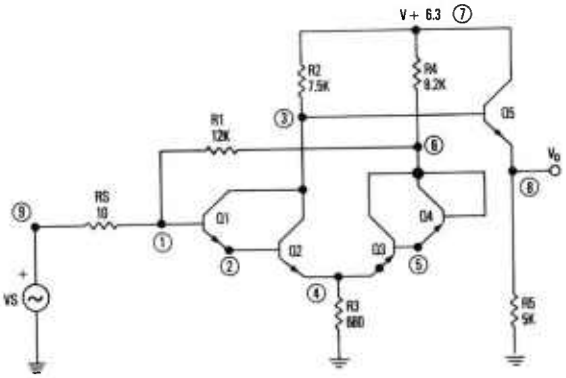


Figure 5. Nine-node integrated preamplifier example circuit.

Convergence.—For linear circuits, the circuit node voltages are obtained by a single matrix inversion; however, for nonlinear (transistor) circuits, BIAS-D iterates to a solution by updating the transistor model parameters after each iteration. It is therefore necessary to determine when the circuit has converged to a proper solution and terminate the iteration process. Ideally, each class of circuit should have its own convergence criterion; practically, however, this is not possible.

The criterion used in BIAS-D is similar to that used in BIAS-M [16]. The criterion uses the square of the node voltage changes from the previous iteration summed over all nodes, that is

$$S_k = \sum_{n=1}^{N} (V_n - V_{n-1})^2 \quad , \qquad (9)$$

where k is the present iteration count, N the total number of circuit nodes, V_n the present node voltage at node n and V_{n-1} the past node voltage at node n. Since S is determined after each iteration, the values for S at the past two iterations, S_{k-1} and S_{k-2}, are also available. If, during any three consecutive iterations, the values of

$$\sqrt{S_k/N} \quad , \quad \sqrt{S_{k-1}/N} \quad , \text{ and } \sqrt{S_{k-2}/N} \qquad (10)$$

are less than 10 μV, then convergence is assumed. If S_k has increased for three consecutive iterations and remains below 1 mV, the iteration process is also terminated with a possible error noted. Other more elegant techniques, such as those reported by Nagel [6] and Freret [9], are possible at the expense of additional speed and memory.

Two other analytical procedures worthy of mention are those that determine small-signal ac gain and input resistance and transient analysis.

Small-signal gain and input resistance.—The method for computing small-signal voltage gain and input resistance used in large computer programs such as BIAS-3 and SPICE requires several complex operations. A dc voltage source is required at the circuit input node. The value of the source must be the same as the circuit's dc quiescent operating point. In a batch-operated environment, this requires an additional computer run. The matrix equation—equation (3)—is solved for the node voltages with unity currents entered into the current vector at the input node locations. The resulting node voltages are then used to determine the voltage gain and input resistance—see equations (11) to (13).

The method implemented in BIAS-D requires only a simple division and does not need an additional computer run. No voltage sources are needed at the input nodes. This method takes advantage of the true matrix inverse available from the matrix-operations ROM. After the dc operating point solution for a given circuit has been determined, the contents of the admittance matrix, Y (which has been inverted to obtain the node voltages), now contains the impedance matrix, Z. Since the admittance matrix was linearized about the dc operating points, the resulting impedance matrix is also linear. This matrix can be used to calculate the input resistance and gain of the circuit.

The transimpedance between an input port (node j to datum node) and an output port (node k to datum node) is

$$Z_{kj} = V_k/I_j \quad , \qquad (11)$$
$$I_n = 0 \quad ,$$
$$n = 1,2, \ldots N \quad ,$$
$$n \neq j \quad ,$$

where Ij is an excitation current. The input impedance at node j is similarly

$$Zjj = Vj/Ij \quad , \qquad (12)$$
$$I_n = 0 \quad ,$$
$$n = 1,2, \ldots N \quad ,$$
$$n \neq j \quad .$$

The transfer voltage ratio (open circuit transfer voltage gain) between any two circuit nodes j and k is found by dividing equation (11) by equation (12) as

$$V_k/V_j = Z_{kj}/Z_{jj} \quad . \qquad (13)$$

Both Z_{kj} and Z_{jj} are available from the dc operating point calculations. In fact, the inverted admittance matrix contains all circuit input resistances and voltage gains. These are easily obtained with an interactive program such as BIAS-D.

Transient analysis.—In a transient analysis simulation, the node voltages must be determined as a function of time. A flow diagram of the transient analysis procedure used in BIAS-D is shown in figure 6. In BIAS-D (BASIC), the only time-dependent element is the capacitor. The voltage across a capacitor with time is given by

$$v = 1/C \int idt \quad . \qquad (14)$$

In computer simulation the value of this integral must be approximated. BIAS-D uses the trapezoidal integration method [17]. With the trapezoidal method a capacitor is represented as a conductance in parallel with a voltage-dependent current source [18]. The time dependence is introduced by loading this conductance and current into the admittance matrix and current vector, solving for the node voltages, and then using these voltages to update the equivalent circuit for each capacitor. The local

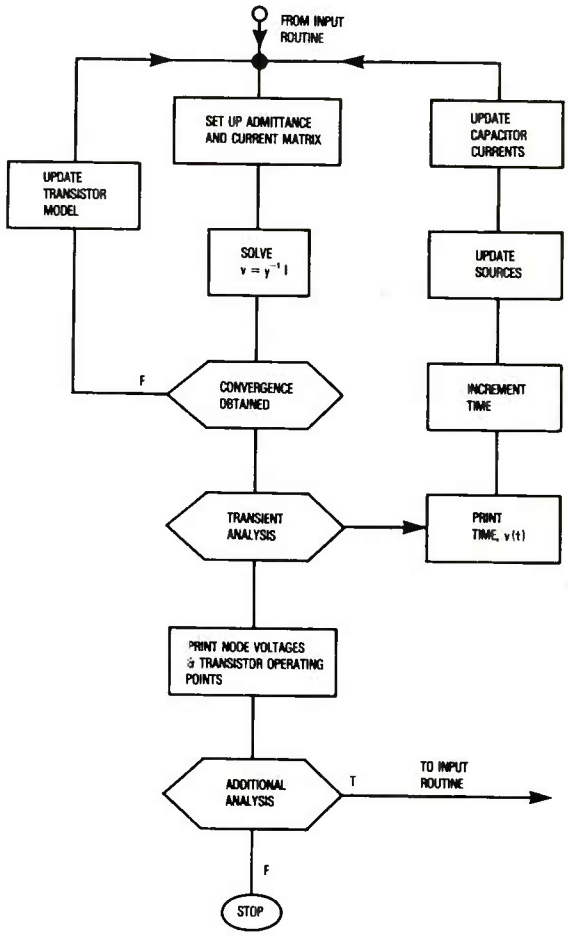


Figure 6. Transient analysis flow diagram for BIAS-D.

truncation error (LTE) associated with the trapezoidal approximation is proportional to the time-step squared [19]. Initially (at t = 0+) this error can be large, and depends on how the algorithm is started. In BIAS-D the dc solution is used for the t_{n-1} solution, and a forced delay equal to the time-step is used for the t_n solution. With this scheme the t used in the LTE calculations is effectively twice the time-step—thereby increasing the truncation errors involved. This procedure was necessary to conserve memory.

2.2.4 Results

The amplifier shown in figure 5 [20] is representative of the size and type of circuit suitable for analysis in BIAS-D (BASIC). The input data and results for a dc analysis of this circuit are shown in figure 2. The source resistor R_s is initially large to determine the quiescent dc operating points, input impedance, and gain. It was subsequently altered to 10 ohms in order to determine the dc voltage transfer curve shown in figure 7. This figure compares the results from BIAS-D, SPICE 1 (using an equivalent transistor model), and actual bench measurements. Results from SPICE agreed to four decimal places with BIAS-D.

The transient response of this circuit was also computed. Capacitors of 1 and 10 pF were added across each transistor collector-base and base-emitter junction, respectively, to represent collector junction and base storage capacitances. Results for a time-step of 50 μs compared closely with SPICE. The computation times on an HP9830A calculator for these analyses on this circuit were as follows:

data input (operator dependent)	2.1 min
circuit restructuring	10.5 s
dc analysis (10 iterations)	2.9 min
dc transfer curve (30 points)	43.2 min
transient analysis (30 time points)	54.2 min

An RCA 3040 integrated wideband amplifier was also analyzed. This represents the maximum circuit size for BIAS-D (15 nodes). Results for a dc analysis of this circuit compared to four significant figures with SPICE for all nodes. The total analysis time, excluding entering data, for a dc analysis was 25 min (6 iterations).

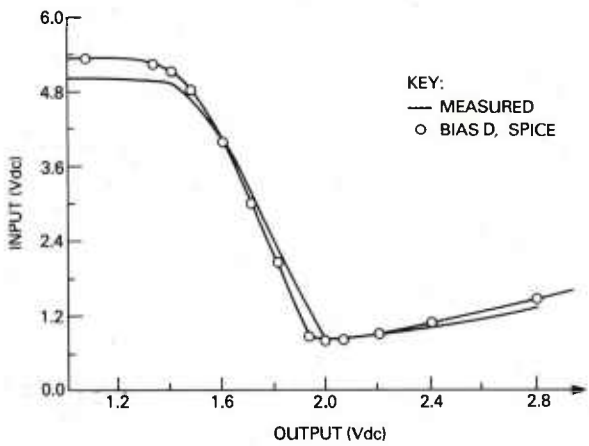


Figure 7. Transfer curve, dc, for preamplifier example circuit, showing BIAS-D, SPICE, and measured results.

These results indicate that the simulation speeds of programmable desktop calculators (at least the HP9830A) are too slow for practical interactive simulation, above the 4 to 5 circuit-node level. BIASL.25 on the HP9825 offers a significant speed improvement (approximately 10:1), but the use of HPL limits its use to the HP9825. Recently available BASIC language calculators such as the HP9845 or the Wang PCS-II have greater memory capabilities and are as fast or faster than the HP9825. These calculators permit practical interactive circuit simulation at the 10- to 20-node level.

A source listing of BIAS-D (BASIC) is given in appendix A.

3. SMALL COMPUTER SYSTEMS FOR CIRCUIT SIMULATION

Small computer systems can be divided into three distinct categories: the minicomputer, the programmable desktop calculator, and the microprocessor. All three systems are capable of circuit simulation. Only the minicomputer and the programmable desktop calculator are included here.

Each of these systems has its own idiosyncrasies and application areas. There are several facets of small computer systems, and in fact all computer systems, which directly concern the development of circuit-simulator codes. These are (1) the computer language, (2) the computer data word format, and (3) the computer speed. Each of these facets is included in this chapter, in which eight computer systems are used for comparison; three of these are minicomputers (an HP2100, a PDP 11/45, and a PRIME 400), two are large computers (an IBM 370/168 and a CDC 6400), and three are programmable desktop calculators (an HP9830A, a Tektronix 4051, and a Wang 2200).

3.1 Minicomputer System

A minicomputer system, as originally conceived, was a small computer system both in size and cost. Minicomputers began to appear in the mid 1960's, primarily as controllers for low-cost original equipment manufacturers (OEM), and have gradually increased in size and speed. Currently, some minicomputers are competitive with the mid-range and even large mainframe computer systems [21].

Figure 8 shows the configuration of a minimum minicomputer system for hosting circuit-simulator programs. In this system, the program is entered into main memory through a punched tape or magnetic tape. The system console is used as a terminal to enter circuit data and print out the results. The addition of a disc would greatly enhance the usefulness of this system. It would enable

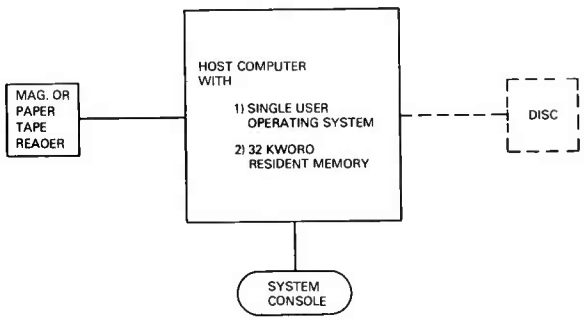


Figure 8. Minimum minicomputer configuration for hosting circuit-simulator programs.

storage of alternative programs and circuit input or output files, as well as allowing the program to be segmented through the use of overlays. The addition of several other features to this minimum system would make it competitive or superior to the larger mainframe computer systems. Figure 9 shows such a system. The magnetic-tape unit allows initial loading of programs, as well as long-term storage of circuit or program files. The system console, used only for monitoring system operation, may not be required. The host computer contains a multi-user operating system which supports several 300- to 9600-baud remote terminals. (A baud is the bit transmission rate; 300 baud is approximately 30 characters per second.) The terminals may be linked to the computer directly (RS-232) or through modems. These terminals may also request copies of numerical or graphical output locally or at the host computer printer/plotter. The host computer also contains virtual memory management. This enables execution of large dynamically allocatable design-aid programs without the need for user segmentation. The system operating speed is enhanced through the use of a small fast-cache memory which speeds up the computer throughput. The size of the required real memory depends on the number of users and their programs' sizes. At least 64 kwords of memory is recommended in a multi-user system.

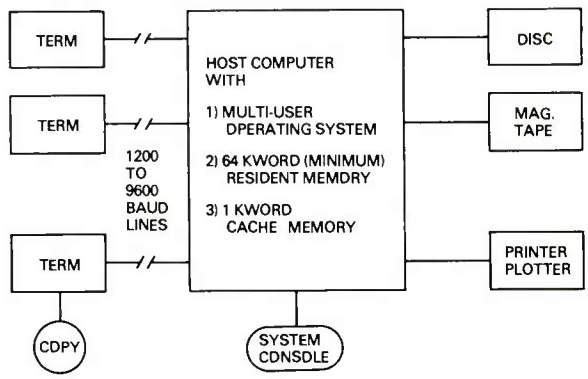


Figure 9. Recommended minicomputer system for hosting circuit-simulator programs.

Minicomputer systems have distinct characteristics that make them different from large computer systems. Some of these characteristics are

1. smaller word size,
2. slower CPU speeds, and
3. use of an ASCII (American Standard Code for Information Interchange) character set in most cases.

Further, minicomputers are economical to run with a single user; their initial cost is low (usually $10K to $500K), and they require no special power or air-conditioning systems. The only characteristics which directly affect the circuit simulator development are the first two.

3.1.1 Computer Languages

There are three basic requirements of a minicomputer language used in developing a circuit simulator: (1) the language should be transportable from one computer system to another with few, if any, software changes required between systems, (2) the language should have a relatively fast execution speed, and (3) the language must be compact and efficient in order to conserve memory. Program transportability is a much greater problem on small computer systems than on the larger systems (such as the IBM's and CDC's). Small systems usually do not have a resident system programmer to modify software, nor is good software or system documentation always available.

The execution speed of a program depends not only on the algorithms used in the program, but also on the execution speed of the language in which the program is written. The potential applications of the program depend not only on the size of the program but also on how efficiently the compiler or interpreter uses memory.

Assembly language.—Assembly language programs can be 1 to 100 times faster than the same code written in FORTRAN and also use less memory than the FORTRAN equivalent, depending on the efficiency of the FORTRAN compiler and the skill of the programmer. Assembly language code is almost never transportable to different computer systems. For this reason, it should not be used as a general language for circuit-simulator development. In special applications, where a short but extremely fast code is desirable, such as LU decomposition, it may be worthwhile to use assembly code written specifically for a particular machine.

BASIC.—Dartmouth BASIC is available on most minicomputer systems. BASIC on most systems is an interpretive language: each line of code is interpreted and executed in the exact sequence that it was written. As a result of this line-by-line interpretation, BASIC is inherently a slow language. For example, on a PRIME 400 minicomputer, BASIC is approximately 14 times slower than FORTRAN IV. BASIC software or hardware decoders can be very compact (4 to 8 kwords of ROM or random-access memory—RAM) and, therefore, are well-suited for programmable desktop calculators. Desktop calculator languages are described in more detail later.

Two versions of BASIC are usually available on minicomputer systems: a popular single-precision version (with 32-bit words) and a double-precision version, DBASIC (with 64-bit words). Because circuit simulators require double-precision word lengths for many calculations [9], the single-precision BASIC cannot be used. DBASIC makes very inefficient use of memory in storing single-precision and integer variables. BASIC (or DBASIC) makes no distinction between integer and real variables. Thus, an integer in DBASIC requires four words of storage when only one word is needed.

In circuit simulators, the use of "string variables" (alphanumeric characters) is essential. Although string-variable features are becoming more widely available, string variables are not a standard subset of BASIC on all computer systems. The use of BASIC or DBASIC for circuit-simulation programs is not recommended except when, as with the desktop calculators, nothing else is available.

FORTRAN IV.—FORTRAN IV is probably the most widely used higher-level computer language. FORTRAN is a compiled language; thus, the speed and efficiency with which this code executes depends on the particular FORTRAN compiler used. Some of the smaller minicomputer systems have three-pass compilers, in which the first pass generates the assembly or object code on a tape or disc, and the

next two passes are needed to convert the assembly code to machine code. As an example of the transportability of FORTRAN IV, a FORTRAN version of BIAS-D was run on three different minicomputer systems and an IBM 370/168 with no changes in the FORTRAN source code [10]; the analytical results were the same on all systems.

3.1.2 Computer Data Word Format

Three types of data formats are usually available in most minicomputer systems: alphanumeric, integer (fixed-point), and floating point. Each has its application. The configuration of each of these data formats determines the magnitude, range, and type of data which can be manipulated or stored in that computer system. To show how differences in these data formats can affect the magnitude and range of allowable numbers, the data formats of five computer systems previously mentioned are compared.

Alphanumeric.—The ability to process alphanumeric characters is extremely important in areas such as circuit simulator input-output languages. Alphanumeric characters are stored in a computer word in an ASCII, BCD (binary coded decimal), or EBCDIC (extended BCD interchange code) code. A cross reference between these codes is given in table 1.

Most minicomputer systems use 7-bit ASCII to represent alphanumeric data. Eight bits are actually used for this code, with the eighth bit used as a parity bit. The parity bit is referred to as either "marked" (1) or "null" (0) parity (the marked parity notation is sometimes referred to as "8-bit ASCII"). This parity bit is important when programs or data are transferred between different computer systems. For example, if the word "NO" were to be stored in a 16-bit word using 7-bit ASCII in marked parity it would be represented as a binary

11001110 11001111 (or an octal 147317) ,

whereas if it were stored in null parity it would be a binary

01001110 01001111 (or an octal 047117) .

TABLE 1. CHARACTER SYMBOL CROSS REFERENCE BETWEEN ASCII, BCD, AND EBCDIC CODES

Symbol	Format (decimal)			Symbol	Format (decimal)		
	ASCII	BCD	EBCDIC		ASCII	BCD	EBCDIC
(space)	32	16	64	A	65	49	193
!	33	[a]	90	B	66	50	194
"	34	[a]	127	C	67	51	195
#	35	12	123	D	68	52	196
$	36	43	91	E	69	53	197
%	37	[a]	108	F	70	54	198
&	38	[a]	80	G	71	55	199
'	39	[a]	125	H	72	56	200
(	40	28	77	I	73	57	201
)	41	60	93	J	74	33	209
*	42	44	92	K	75	34	210
+	43	48	78	L	76	35	211
,	44	27	107	M	77	36	212
-	45	32	17	N	78	37	213
.	46	59	75	O	79	38	214
/	47	17	97	P	80	39	215
0	48	10	240	Q	81	40	216
1	49	01	241	R	82	41	217
2	50	02	242	S	83	18	226
3	51	03	243	T	84	19	227
4	52	04	244	U	85	20	228
5	53	05	245	V	86	21	229
6	54	06	246	W	87	22	230
7	55	07	247	X	88	23	231
8	56	08	248	Y	89	24	232
9	57	09	249	Z	90	25	233
:	58	00	122				
;	59	63	94				
<	60	58	76				
=	61	11	126				
>	62	47	110				
?	63	[a]	111				
@	64	[a]	124				

[a] *Not permitted*

It is possible to convert from null parity to marked parity (or vice versa) by adding (or subtracting) an octal 100200 to each 16-bit alphanumeric word. This cannot be done in ANSII (American National Standard for Information Interchange) Standard FORTRAN but is easily done in assembly code.

In the HP2100 [22], PDP 11/45 [23], and the PRIME 400 [24] minicomputer systems, alphanumeric characters are represented in ASCII format and stored in a 16-bit word as follows:

```
15              7 ·             0
[p][character 1][p][character 2]
```

where p indicates the parity bits. Two characters can be stored in a single word. If a single character is to be stored, it is right justified for the PDP 11/45 and left justified for the others. The other character is filled with ASCII blanks (different from zeros).

In the HP2100 and PDP 11/45, null parity is used, whereas, on the PRIME 400, marked parity is used. The CDC 6400 system [25] uses a 6-bit BCD code to represent alphanumeric characters. Ten 6-bit characters are packed into a single CDC 60-bit word as

```
59                                          0
[c1][c2][c3][c4][c5][c6][c7][c8][c9][c10]
```

If less than ten characters are to be represented, they are left justified and the remaining characters filled with BCD blanks.

The IBM 370 system uses 8-bit EBCDIC to represent alphanumeric characters. EBCDIC is merely an extension of the 6-bit BCD code and permits 256 characters rather than the 56 allowed for BCD. On the IBM 370 system [26], four 8-bit EBCDIC characters are packed into a 32-bit word as follows:

```
31                                  0
[char 1][char 2][char 3][char 4] .
```

Again, if less than four characters are to be represented, they are left justified with the remaining characters filled with EBCDIC blanks.

Integer number.—The integer is used to represent numbers which do not require decimal fractions. In most computer systems an integer is represented by a single computer word. In the minicomputer systems under discussion, this is a 16-bit word, with the highest-order bit being the sign bit as

```
15                          0 ·
[s][       number           ]
```

This can be used to represent an integer number from -2^{15} (–32768 decimal) to $2^{15} - 1$ (32767 decimal) including zero. The IBM and CDC systems have a similar representation with the IBM 370 using a 32-bit word (sign plus 31 number bits) and the CDC 6400 using a 60-bit word (sign plus 59 number bits). These larger word sizes allow a much greater range of integer numbers. However, in circuit simulation this additional range is almost never required.

Floating-point numbers.—A floating-point number is represented by a mantissa (fraction) and a characteristic (exponent). The fraction determines the accuracy of the floating-point number and the exponent to some base determines the range. Both base 2 (binary) and 16 (hexadecimal) are used in minicomputer systems. There are basically four types of floating-point numbers: single precision (real and complex) and double precision (real and complex). Floating-point hardware is available as an option on most minicomputer systems. This option always includes the single-precision hardware, sometimes the double-precision hardware, but never the complex floating-point hardware.

Single-precision real floating-point number.—Usually in minicomputer systems two words are used to represent a single-precision floating-point number. Table 2 shows the single-precision floating-point number representation for the HP2100, the PDP 11/45, the PRIME 400, the IBM 370/168, and the CDC 6400. Note that in each case this representation is different. The resulting precision and range in each case is given as shown in table 3. Note that the PDP 11/45 system attains seven digits of precision with the same number of bits as the HP2100 and PRIME 400. This is done by using "hidden-bit normalization" which assumes that the normalized highest-order bit is always a 1 (unless the exponent is zero) and is, therefore, unnecessary. This gives an effective precision of 24 bits in the fraction. The large range of the IBM number is attained by using the hexadecimal number system rather than binary to represent the exponent ($16^{64} = 10^{78}$, whereas $2^{64} = 10^{18}$).

Single-precision complex floating-point number.—Minicomputer systems which support single-precision floating-point arithmetic usually support single-precision complex floating-point arithmetic.

TABLE 2. COMPARISON OF SINGLE-PRECISION DATA FORMATS

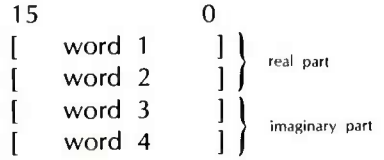
Type of system	Format[a]
HP2100	15 0 word 1 [s][fraction] 15 7 0 word 2 [fraction][exp][s]
PDP 11/45	15 6 0 word 1 [s][exp][fraction] word 2 [fraction]
PRIME 400	15 0 word 1 [s][fraction] 15 7 0 word 2 [fraction][exp]
IBM 370/168	0 8 31 [s][exponent][fraction]
CDC 6400	59 47 0 [s][exponent][fraction]

[a]*s = sign*

TABLE 3. PRECISION AND RANGE COMPARISONS: SINGLE-PRECISION NUMBERS

Computer	Precision (decimal digits)	Range (decimal)
HP2100	6	10^{-38} to 10^{37}
PDP 11/45	7	10^{-38} to 10^{37}
PRIME 400	6	10^{-38} to 10^{37}
IBM 370/168	6	10^{-77} to 10^{76}
CDC 6400	12	10^{-307} to 10^{306}

However, this complex arithmetic is usually done in software (even on the large computers). A complex floating-point word is represented by two single-precision floating-point numbers: the first number is the real part of the complex word, and the second is the imaginary part. Since the complex number is actually two real numbers, the magnitude and range is the same as for the single-precision real floating-point numbers. On 16-bit per word computers, four 16-bit words are required for a complex number as follows:



On the 32- and 60-bit machines (IBM 370 and CDC 6400), only two words are required, the first for the real part and the second for the imaginary part.

Double-precision real floating-point number.—The configuration and execution speed of the double-precision numbers in minicomputer systems are very machine dependent. Double-precision hardware or firmware,* if available, is usually an option. If executed in software, double-precision arithmetic must be written in assembly or machine code and is therefore several times slower than its hardware counterpart. Table 4 gives the double-precision word configuration for the HP2100, the PDP 11/45, the PRIME 400, the IBM 370, and the CDC 6400. The resulting precision and range for each of these systems are given in table 5.

Table 5 shows that the precision and range of double-precision numbers can vary considerably between computer systems—more so than the single-precision numbers.

It is the size of this double-precision word that limits the maximum circuit size. It has been shown that with a well-conditioned set of equations, round-off error can reduce the number of significant digits by a factor

$$1 + 2(\log N),$$

where N is the number of circuit nodes [6]. Three to six significant digits are required for circuit simulation. For all computers listed above, it should be possible to solve a 50-node equation, and on all but the HP2100, a 1000-node equation. Techniques

**Firmware is used here to mean software which has been implemented in microcode or read-only memory.*

TABLE 4. COMPARISON OF DOUBLE-PRECISION DATA FORMATS

Type of system	Format	
HP2100		15 0
	word 1	[s][fraction bits]
	word 2	[fraction bits]
		15 7 1 0
	word 3	[fraction][exp][s]
PDP 11/45		15 6
	word 1	[s][exp][fraction]
	word 2	[fraction]
	word 3	[fraction]
	word 4	[fraction]
PRIME 400		15 0
	word 1	[s][fraction]
	word 2	[fraction]
	word 3	[fraction]
	word 4	[exponent]
IBM 370/168		0 8 31
	word 1	[s][exponent][fraction]
	word 2	[fraction]
CDC 6400		59 47 0
	word 1	[s][exponent][fraction]
	word 2	[s][exponent][fraction[a]]

[a] *LSB*

TABLE 5. PRECISION AND RANGE COMPARISONS: DOUBLE-PRECISION NUMBERS

Computer	Precision (decimal digits)	Range (decimal)
HP2100	10	10^{-37} to 10^{38}
PDP 11/45	17	10^{-37} to 10^{38}
PRIME 400	13	10^{-9865} to 10^{9866}
IBM 370/168	14	10^{-77} to 10^{76}
CDC 6400	27	10^{-307} to 10^{306}

for minimizing round-off error, such as pivoting [27] or those developed by Freret [9,28,29], can be used to increase this node capability. Since the interest here is at the 30- to 50-node level, the use of these techniques is not necessary.

Double-precision complex floating-point number.—Although most minicomputer systems offer double-precision floating-point arithmetic in hardware or firmware, double-precision complex arithmetic is not available. Double-precision complex arithmetic must be done as a software subroutine call. The primary disadvantage is the resulting speed of operations. Implementation of double-precision arithmetic on software is 10 to 100 times slower than implementation on hardware or firmware. Sometimes, it is possible to implement this arithmetic into a writable control store (WCS) or microcode (usually an option) which is essentially a programmable read-only memory (PROM). For example, to execute a single- or double-precision software complex divide represented as

$$C = A/B = (A_R + jA_I)/(B_R + jB_I) , \quad (15)$$

where R indicates the real part of the complex and I the imaginary part, the resulting real and imaginary parts of C must be computed separately as

$$C_R = (A_RB_R + A_IB_I)/(B_R^2 + B_I^2) , \quad (16)$$

$$C_I = (A_RB_I - A_IB_R)/(B_R^2 + B_I^2) . \quad (17)$$

These operations require six double-precision multiplies, two divides, two adds, one subtract, and one store.

3.1.3 Computer Speed

The speed of operation of a minicomputer system depends on several factors:

1. the configuration of the system,
2. the language used,
3. the type of arithmetic executed and mode of implementation (software, firmware, or hardware),
4. type of memory (core, bipolar, or MOS) and its access speed, and
5. CPU clock speed.

All the above factors determine the execution time of a particular program.

The system configuration affects the overall speed of each job. If a single user is running on a multi-user system he is penalized in actual run time (not necessarily in CPU time) because of overhead. If several users are on a multi-user system and the computer becomes compute- or memory-bound, all users will be penalized in overall run time. The language used and how this language is managed in the particular computer system can greatly affect the run time. An interpretive language will always be relatively slow. The speed of a compiled language is determined in part by how efficiently the compiler-generated machine code executes; this efficiency depends in turn on the efficiency of the basic machine instruction set. The type of arithmetic being executed also can affect the total run time. If the execution times on a PRIME 400 of an assembly ADD instruction are compared for an integer ADD, a single-precision floating-point ADD, and a double-precision floating-point ADD, they would be in the following ratios.

Arithmetic	Speed ratio	Implementation
Integer	1	hardware
Single-precision floating point	7	firmware
Double-precision floating point	9	firmware

These ratios indicate that, on a PRIME 400, integer arithmetic should be used wherever possible. This is generally true for all computer systems.

It is difficult to compare the overall speeds of different computer systems since, as was just mentioned, there are many variables which affect this speed. To compare computer systems for use by circuit simulators, the best comparison is to run a circuit-simulator program. Such a speed comparison of four computer systems (an HP2100, a PDP 11/45, a PRIME 400, and an IBM 370/168) is given in section 4. This comparison is made using a FORTRAN version of BIAS-D which runs on all systems with no source code changes.

3.1.4 Computer Memory Configuration

Another basis for comparison of minicomputers is the configuration of the memory. All present-day computer systems use two types of memory storage: small, rapid-access, relatively expensive, resident-memory storage, and large, slow-access, disc- or tape-memory storage. The procedures for managing these two types of memory can greatly affect the operation of the computer system.

There are two basic memory management schemes: real memory management and virtual memory management. Real memory management restricts the user to a segment of the total available memory (usually 32 kwords). Within this segment, the user can control his own memory management through the use of overlays to disc memory. Virtual memory management [30], in theory, allows the user the advantages of both types of memories. That is, it permits a large memory to be addressed at access times of the fast memory. In a virtual memory, "pages" are moved in and out of resident memory as required. With this memory system, overlaying of program segments is not necessary. This results in a program which is easily transportable to other virtual or large-memory computer systems. In the computer systems compared previously, only the PRIME 400 minicomputer and the IBM 370/168 have virtual memory management.

3.2 Programmable Desktop Calculator

Programmable desktop calculators began to appear in the mid-1960's, at about the same time as the minicomputers. The development of these calculators was relatively independent of the minicomputers. It has been only since the appearance of the "super" calculators such as the HP9830, Wang 2200, and Tektronix 4051 that the minicomputers and calculators could speak a common language—BASIC. Although the computing power of desktop calculators approaches or exceeds that of small minicomputers, there are still definite differences in these systems. Some of the distinguishing features of the desktop calculators are as follows.

1. The keyboard is an integral part of the computer.
2. The computer language is permanently stored in the machine either in hardware or firmware (ROM).

3. The programming language is an interpretive language (at present).
4. The desktop calculator comes as a turnkey* system.

Desktop calculators, unlike minicomputers, do not provide a wide choice in input language, data word size, or computational speed.

3.2.1 Calculator Language

In the presently available programmable calculators, the programming language is not an option. It is permanently stored in the calculator. There are presently only two minicomputer-compatible languages, BASIC and APL, available on these calculators. Other languages are hybrids between BASIC and an assembly language. For example, HPL (Hewlett Packard Language) is used on the HP9825. All calculator languages are presently interpretive languages. This means that each line of the program is interpreted and executed line by line exactly as it was written. Calculator languages therefore are relatively slow when compared with compiled languages.

The BASIC language implemented in the desktop calculators is a superset of Dartmouth BASIC. Several features have been added which greatly enhance the usefulness of BASIC. String-variable operations (comparable to alphanumeric or Hollerith characters in FORTRAN) are available either in factory-added hardware/firmware or as a user-added plug-in ROM. Other plug-in ROM's allow matrix inversion with a single line of code in a tenth the time required in software. Also available are other features, such as bit and byte manipulations, data packing and unpacking, or variable data word lengths.

3.2.2 Calculator Data Word Format

Two types of data word formats are generally available on the programmable calculators: string variable and numerical.

**A turn-key system is a system that is ready to use as soon as it is delivered and turned on.*

The string-variable word is used to store or manipulate alphanumeric data. Seven-bit ASCII is used to represent these data, and a single character requires eight bits as in the minicomputers. The Wang 2200 system [31] permits storing or "packing" of numerical data into string-variable arrays, which enables high-density data storage.

The numerical data word is used to store integer and floating-point data. The data word does not differentiate between integer and floating-point numbers. Except for dimensioned variables in the HP9830, all numerical data require four 16-bit words for storage and arithmetic operations. This gives 13 to 14 decimal digits of precision and a range of 10^{-99} to 10^{99} for the HP9830A [32] and the Wang 2200 and 10^{-306} to 10^{305} for the Tektronix 4051 [33]. The HP9830A permits specifying full-precision, split-precision, and integer-precision words in dimensioned variables. These require 64, 32, and 16 bits, respectively, for storage. The resulting precision and range for each of these words is affected by the shorter word length.

3.2.3 Calculator Speed

The computational speed of the desktop calculator is not as dependent on the calculator configuration as is that of the minicomputer. The calculator systems are always single-user systems with hardwired or firmware interpretive programming languages. In some cases, the addition of special-function ROM's could significantly change the computational speed of a particular set of operations, but in most cases the calculator speed is only a function of the clock cycle time.

As a comparison of the relative speeds of these calculators, a simple loop containing a multiply and divide operation was executed 10,000 times. The resulting normalized speeds were as follows.

Calculator	Normalized speed
HP9830A	1.0
Wang 2200	0.48
Tektronix 4051	0.45

As an indication of the speed of these calculators compared with a minicomputer system, the execution speed of a FORTRAN version of BIAS-D running on a PRIME 400 minicomputer is 500 to 600 times faster than a similar BASIC version of BIAS-D running on an HP9830A desktop calculator.

4. CIRCUIT SIMULATION ON MINICOMPUTERS

Present-day minicomputer systems have or exceed the capabilities of the large computer systems of five or ten years ago. Even so, certain limitations in both minicomputer hardware and software must be considered for present-day circuit-simulator development. Hardware aspects of the minicomputer circuit simulators were presented in section 3. This section is oriented toward the software aspects of circuit-simulator development.

A large simulator program, such as SPICE, can be converted to run on a minicomputer system. Later in this chapter results are given from SPICE2, run on a PRIME 400 minicomputer.* The conversion of these programs from the larger computer systems to minicomputer systems is not always practical, however. Many of these small systems have limitations (such as 32-kword program boundary limits) that make this approach difficult and uneconomical.

Another program in which this conversion was successfully done was Mini-MSINC [2]. Mini-MSINC, developed for an HP2100 minicomputer system with a DOS III operating system, was derived from TIME [34], SINC [12], and MSINC [35], all developed for large computer systems. To fit Mini-MSINC into the 32-kword memory of the HP2100, it was necessary to overlay memory through five overlay segments on disc and to extensively modify the common array allocation with linked lists [30]. Mini-MSINC is probably the most widely used minicomputer circuit-simulator program at this time. It can do a dc or transient analysis of MOS circuits containing over 100 nodes and 100 active devices. Although Mini-MSINC has been restricted to the analysis of MOS circuits, it is presently being updated to analyze bipolar transistors and to perform ac and statistical analysis.*

Simulation speed is an important consideration in choosing the type of simulation program to be used (batch mode or interactive mode). In order to make this choice, it is necessary to determine the computational speeds of different minicomputer systems using a circuit-simulator program. This can best be done with a circuit-simulator program that is compatible with all systems being evaluated. Comparisons of both the speed and memory requirements of several computer systems using BIAS-D are given in section 4.4.

The architecture of the circuit-simulator program is influenced by three basic areas of software development. These are

1. simulator operation,
2. simulator speed, and
3. simulator memory requirements.

Interaction between each of these areas represents trade-offs which can affect one or both of the other areas. For example, a software routine that could greatly increase the simulation speed may also require significantly more memory. In the large computer systems these trade-off problems are less significant than in minicomputers. The large computer systems usually sacrifice readily available memory for speed. On many of the larger systems there is no penalty for using this additional memory. On the minicomputer systems, additional memory is not always available and is at a premium. Trade-offs must be made which hold program memory requirements within a given bound at the expense of either the simulator operation or simulator speed.

4.1 Simulator Program Operation

There are basically two types of circuit-simulator operation: batch and interactive. Both are software oriented.

In the batch mode, the program or data are entered into the computer through a "hopper" which is linked directly to the computer. Results are

**M. Payne, PRIME Computer Corp., private communication.*

**R. W. Dutton, Stanford University, private communication.*

returned on a high-speed line printer. No interaction with the program is possible in the batch mode.

Variations of the batch mode include the remote batch and interactive batch modes. The remote batch mode is similar to the batch mode, except that the hopper and printer are remote, linked by high-speed communication lines or modems (4,800 to 120,000 baud). In the interactive batch mode, an indirect interaction with the program is possible through the use of disc files and an editor. Here, a low-cost terminal is tied to the host computer, usually through standard telephone lines and a low-speed modem (110 to 1200 baud). Figure 10 shows the input processing portion of a circuit simulator using the interactive batch mode. Here a previously generated circuit data file is entered into the program. The data are processed and checked for errors. Data errors terminate the job. If the circuit is error free, the circuit is set up and analyzed, and the resulting output is stored for future printing or plotting. If a circuit change is required or a new circuit file is to be generated, an editor must be used, as shown in figure 10. Once the circuit file has been updated or generated, it is stored in memory (disc) and the circuit simulation restarted.

Circuit simulation using the interactive batch mode of simulation, although superior to batch, is still awkward. This is especially true for small minicomputer systems which have crude editors. SPICE1 and SPICE2, as they were originally written, were intended for use as batch mode simulators. Some modified versions of these programs, such as ISPICE [36], are interactive batch oriented.

The interactive mode simulator is significantly different from the batch mode circuit simulator, which has little or no interaction. The interactive simulator allows direct interaction with the program. An input flow diagram of an interactive simulator input processor is shown in figure 11. Here the data are entered, processed, and checked for errors one line at a time. Syntax errors are immediately flagged, allowing the data to be reentered. When the data entry is complete, the circuit

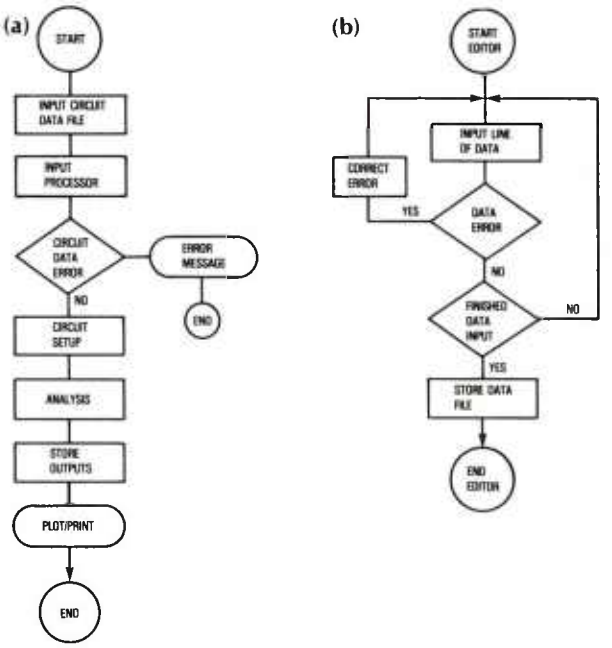


Figure 10. Flow diagram for interactive batch simulation: (a) input processor and (b) typical editor.

is set up and analyzed by the computer. An interrupt flag, set by a predetermined keyboard entry, can stop the analysis at any time and return to the input portion of the program. Outputs are either printed or graphically displayed on the terminal as they are computed. At the end of an analysis, the program may be terminated or returned to the input processor. To better illustrate the flexibility of the interactive simulator, the command instruction set from BIAS-D is given in table 6, along with a brief description of each command.

These commands can be used at the end of any analysis and allow freedom in the simulation procedure. This enables the type of analysis or circuit modification to be determined pending the outcome of the previous analysis.

Although an interactive circuit simulator is desirable, it is not always practical. If a computer system is so slow that the engineer or designer must wait several minutes or hours for the simulation results, and then respond to these results, an interactive simulation should not be used. In this case a batch simulator is best. The computational speed

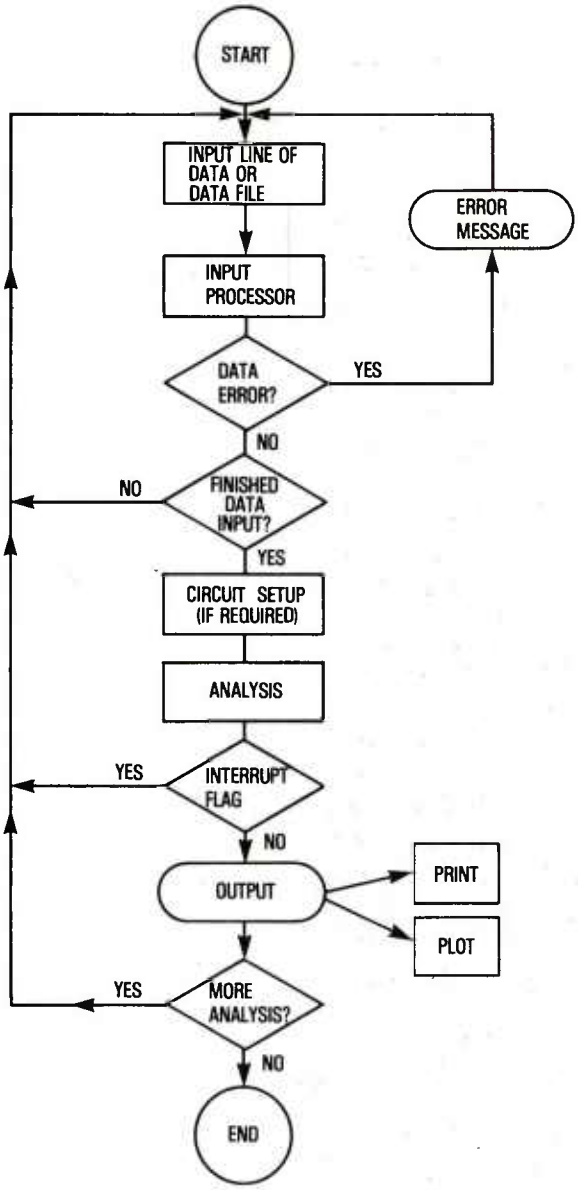


Figure 11. Flow diagram for an interactive circuit-simulator input processor.

breakpoint between the interactive simulator and batch simulators is discussed in section 4.4.

4.2 Speed-Dependent Simulator Software

The relative speed of a circuit simulation program is very dependent on the algorithms used. Some of the more significant techniques for improving simulation speed are discussed here. The solution of the matrix equation $YV = I$ for the circuit node voltages represents a significant portion of the memory and speed required for a simulation. Zero checking, node reordering, and sparse decomposition are three techniques which can be used to speed up this solution. BIAS-D was used to evaluate the effect of these techniques and others to be described subsequently.

TABLE 6. COMMAND INSTRUCTION SET FROM BIAS-D

Command	Description
.AC	Initiates ac analysis
.ALTER	Permits altering or sweeping element values
.END	Terminates present circuit analysis and initializes memory for new circuit
.INSERT	Permits insertion of any circuit element or elements (including models)
.LOAD	Permits loading of circuit data from a disc file
.PRINT	Prints present circuit topology
.SAVE	Saves present circuit on disc file
.TEMP	Permits analysis of circuit at temperatures other than 27 C
.TRAN	Initiates transient response analysis

An initial test version of BIAS-D did not use any speed- or memory-improvement techniques. The matrix equation was solved with a standard double-precision LU decomposition with forward and backward substitution [6]. This will be referred to as the standard version of BIAS-D.

4.2.1 Test Circuits and Procedures

Four test circuits were used to compare the analytical speeds of BIAS-D modifications described in this section. These test circuits were all

modifications of the same test circuit used in section 3 to test the BASIC version of BIAS-D. Diagrams of these circuits and input listings are given in appendix B (fig. B-1 through B-4). The initial circuit (CKT10) is a 9-node, 5-transistor integrated preamplifier circuit. Capacitors were added across the collector-base and base-emitter junctions of each transistor to represent the transistor junction capacitances. CKT10 does not include any bulk resistors, but the other three circuits were obtained from CKT10 by successively adding resistors to the base (CKT11), collector (CKT12), and emitter (CKT13) of each transistor in this circuit. The element count, number of nodes, and sparsity of each of these circuits is given in table 7.

TABLE 7. COMPARISON OF TEST CIRCUITS

Circuit Name	Nodes	Element count				Percent sparsity
		R	C	V	Q	
CKT10	9	5	10	2	5	55
CKT11	14	10	11	2	5	70
CKT12	19	15	11	2	5	74
CKT13	24	20	11	2	5	79

These circuits were used in all subsequent speed comparison tests. The computational speed tests were conducted with 101 timepoints of a transient simulation run on a PRIME 400 minicomputer with a PRIMOS IV (revision 13) operating system. The input signal for all tests was a single unity amplitude voltage pulse at node 9 of the test circuits. A plot of both the input pulse and output waveform for CKT13 is given in figure 12. A single test run determined the CPU time per iteration for each of the four test circuits by dividing the total CPU time by the number of iterations. The final analysis times for each run were determined by averaging these CPU times for three runs. The final data plots were obtained by fitting these results to a least square fit. A semilog fit of the nodes and log time per iteration produced results with the best fit (largest correlation coefficient). The equation for this fit is of the form

$$\text{time/iteration} = B \times 10^{m \times \text{Nodes}}, \quad (18)$$

where 10^B is the y-axis intercept and m is the slope. The objective of the following tests is to minimize both this intercept and slope. The standard version of BIAS-D was successively modified to include the six speed and memory enhancements described subsequently (BIAS-T1 to BIAS-T6).

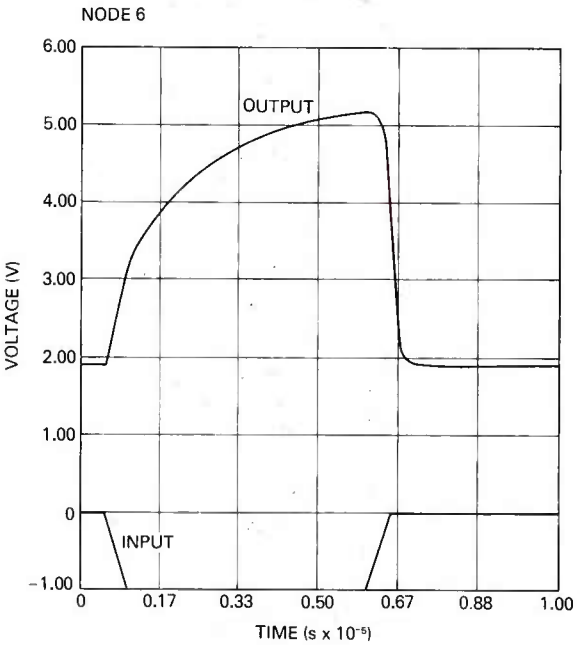


Figure 12. Input pulse and pulse response for CKT13 using BIAS-D.

4.2.2 Zero Checking

The first modification to BIAS-D (BIAS-T1) involved a simple modification in the matrix inversion process. This modification simply checked the value of an admittance matrix location for zero before an operation was performed. An operation is defined as a multiplication and a subtraction (as xy − z) in the decomposition process. Both x and y can be checked for zero; if a zero is found, either that operation or an entire row (or column) of operations may be omitted. This procedure was used in early versions of BIAS-3 [11] and SLIC [4]. Since, in circuit simulation, the admittance matrix is always sparse (i.e., more than 50 percent of the entries are zeros), significant time could be saved at

the expense of checking each entry for a zero value. On the PRIME 400 minicomputer there is approximately a 15:1 CPU time saving between computing a single zero check and a single operation (in double-precision arithmetic). The order in which the circuit nodes are numbered can also determine whether a single operation or an entire row of operations is skipped. A Markowitz reordering [37] scheme can be used to determine a near optimum circuit node ordering. Figure 13 compares the CPU time per iteration versus circuit nodes for the standard test program with no modifications, and BIAS-T1 with the zero checking modifications. Several results are given here: those from (1) the standard version of BIAS-D, (2) BIAS-T1 with zero checking with random circuit-node numbering, and (3) BIAS-T1 with the near optimum node ordering as determined from a Markowitz reordering scheme. The Markowitz scheme was not actually implemented in the test program at this time, but was used only to determine the new node orders. The circuit nodes were renumbered according to this order before being entered as input data. All results were obtained from a 101-point transient analysis of the circuits as described in section 4.2.1.

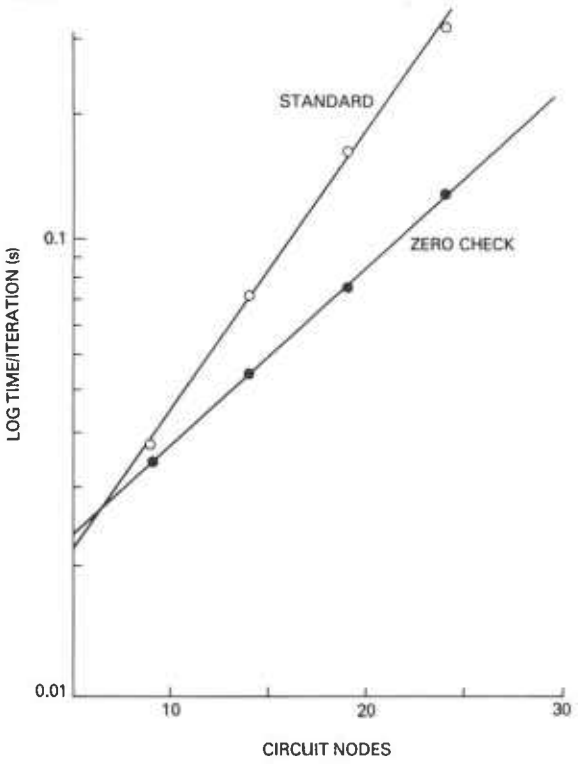


Figure 13. Speed versus circuit nodes for BIAS-D (PRIME 400) using zero test compared with standard version using test circuits.

The results in figure 13 indicate that zero checking is always faster than the standard method for any arbitrary node order. It also indicates that there can be a noticeable difference in analytical speeds owing to the manner in which circuit nodes are numbered, unless a node reordering scheme is used.

4.2.3 Node Reordering

If the Markowitz reordering algorithm is implemented in the simulator program as part of the setup procedure, then (as indicated in the previous section) the analysis speeds will no longer be dependent on the operator-assigned node ordering. Markowitz reordering is used in BIAS-N, which is a later version of BIAS-3 [11] and SPICE2 [6]. Figure 14 shows a comparison of the speeds of the test circuits with the Markowitz reordering scheme actually implemented in BIAS-D (BIAS-T2) and also the best and worst cases from the test program with only zero checking implemented (fig. 13). It should

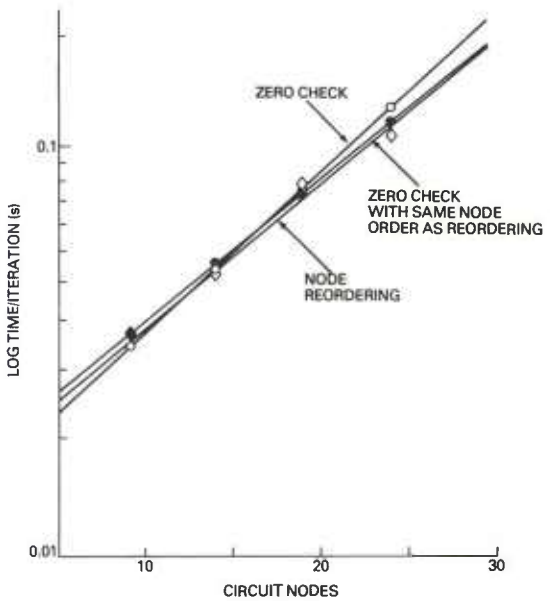


Figure 14. Speed versus circuit nodes for BIAS-D (PRIME 400) using node reordering.

be noted that the node reordering scheme implemented here does not always generate an optimal node order. This can be seen in figure 14 for the 19-node speed data (CKT12). The "optimal" node order generated for this circuit is actually worse than the original node order (discussed again in sect. 4.2.4). Figure 14 indicates that there is only a minimal, if any, speed penalty in the analysis time for using the reordering scheme; however, additional software and memory are required to implement this reordering scheme. The cost of this additional overhead is given in section 4.3.

4.2.4 Sparse Matrix Decomposition

If the matrix LU decomposition process is set up such that "pointers" indicate the matrix location of the next nonzero value for each operation, the time required for a zero check can be eliminated. This pointer system would also permit storage of only these nonzero terms. In order to set up this pointer structure it is necessary to perform a "mock decomposition" of the admittance matrix. This mock decomposition again requires additional software and memory. BIAS-D was modified (BIAS-T3) to include this sparse decomposition. This decomposition process includes a Markowitz reordering algorithm similar to that used in BIAS-T2. However, the reordering scheme incorporated in the mock decomposition results in a more efficient reordering than in BIAS-T2 because the matrix "fill-ins" are counted during the mock decomposition process. These fill-ins were not determined in BIAS-T2 since the mock decomposition was not required in that matrix reduction. Sparse matrix storage was not implemented in BIAS-T3. Figure 15 shows a comparison of the analytical speed on a PRIME 400 of the sparse decomposition version of BIAS-D with that of the several previous versions. As can be seen in this figure, the sparse decomposition process represents a significant increase in speed. Some of this speed increase can be attributed to the more efficient reordering just mentioned.

4.2.5 Processed Element Storage Array

Another apparent speed-up procedure, currently used in several circuit-simulator programs

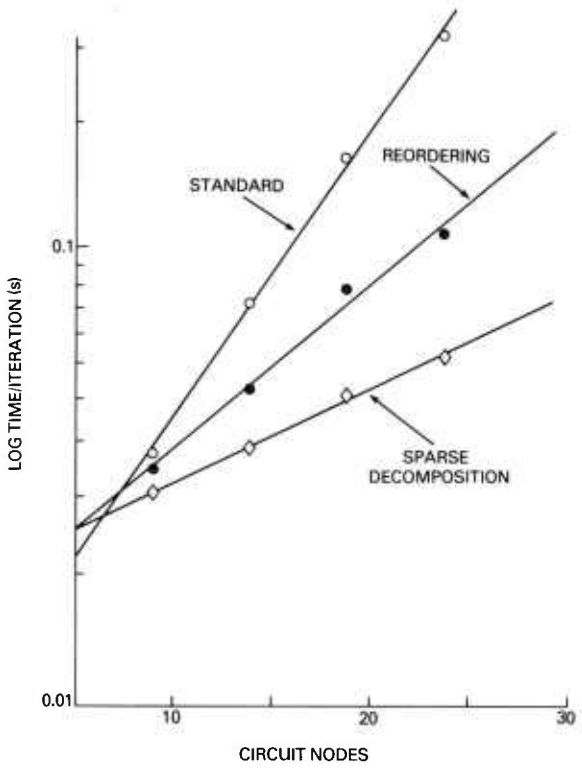


Figure 15. Speed versus circuit nodes for BIAS-D (PRIME 400) using sparse decomposition compared with node reordering.

[2,4,6,12], is the storage of a processed-element array. This array contains the present conductance values to be added to the admittance matrix. The equations used to compute this array value are different for each element and are given for linear resistors, capacitors, and inductors as

$$\text{Resistors} \qquad (1/R)T_C \tag{19}$$

$$\text{Capacitors} \qquad (2C/\Delta)T_C \tag{20}$$

$$\text{Inductors} \qquad (\Delta/2L)T_C \tag{21}$$

where R is the resistance value, C the capacitance value, L the inductance value, Δ the present time-step, and T_C the temperature multiplication factor ($T_C = 1$ at 27 C). For resistors, this array need only be computed once for each analysis. This is also true for capacitors and inductors unless a "time-step control" [6] is used in the transient analysis

procedure. In this case Δ may vary with time, requiring updating of the processed array during a transient analysis.

BIAS-T3 was modified to include this double-precision processed-element array (BIAS-T4). Equations (19) through (21) were used to load this array once for each transient simulation (BIAS-T4 does not use time-step control). Figure 16 shows the results from BIAS-T4 for transient simulations of the standard test circuits using the PRIME 400 minicomputer. Also plotted in figure 16 for comparison are the results without this array. Surprisingly, for CKT13 there is less than a 0.5-s speed improvement in the 101-point transient analysis because of this array. After a second look, however, we can see that this small improvement is all that should be expected. The approximate PRIME 400 assembly language instruction speeds are 26 μs for a double-precision multiply, 33 μs for a divide, and 2.5 μs for a store. Test circuit CKT13 contains 10 capacitors and 20 resistors. Using equations (19) and (20) this represents a time saving of 1.2 ms for resistors and 1.0 ms for capacitors at each timepoint in the analysis. For the 101-point analysis this represents a total of 0.22 s, which is the approximate difference shown in the figure. If the analysis is done at a temperature other than 27 C, the time savings due to this array are increased. This is because the calculation of T_C, in equations (19) to (21), involves evaluation of a second-order polynomial. This same test using BIAS-T3 and BIAS-T4 was run again at 50 C. These results are also shown in figure 16. Even at 50 C, this does not represent any significant speed savings.

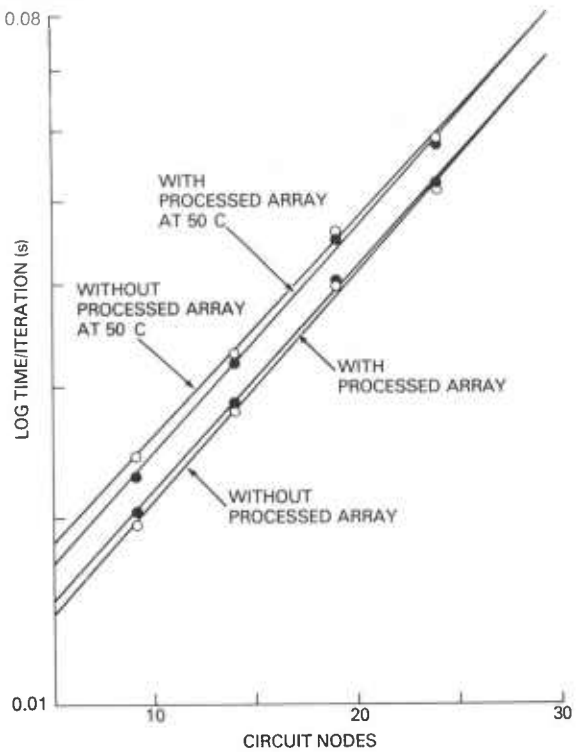


Figure 16. Speed versus circuit nodes for BIAS-D (PRIME 400) with additional processed-element storage array.

If sparse matrix storage is used, then it is also desirable to store not only the processed-element array, but also the address location in the admittance matrix where this array is to be added. Depending on the implementation of the sparse array storage, resistors, capacitors, and inductors could require from 2 to 4 address locations for each element [2,6,38]. If transistors are included, they could require from 6 to 18 locations. Because of the poor improvement in speed using the processed-element array, it was not expected that any significant improvement could be achieved by storing the address locations in Y for this array. The processed-element array was therefore not implemented.

4.2.6 Summary of Speed-Improvement Techniques

Each of the above techniques (zero check, reordering, sparse decomposition, and processed-element array) reduces analytical times but requires additional overhead. The circuit setup time was not included in the previous analysis times, mainly because this was only done once for each circuit. Memory requirements due to the added software and the pointer storage arrays are overheads which must also be considered. Table 8 summarizes the speed-improvement techniques for each modification of BIAS-D. Table 9 shows the results of these improvements. Included here are the additional lines of FORTRAN code required, the increase in compiled program memory requirements, the increase in memory due to added storage arrays, the

net additional memory used or saved (negative memory indicates a savings), and the relative speed improvement of each modification at the 24-node level. Note that the total memory requirement for BIAS-D is approximately 25k decimal words (including system routines, FORTRAN library, and graphics). Table 9 shows that the addition of any of these speed-improvement techniques, except the processed-element array (BIAS-T4), does not require a significant amount of memory relative to the speed improvement gained.

TABLE 8. SPEED- AND MEMORY-IMPROVEMENT ALGORITHMS USED IN COMPARISONS

Algorithm	Includes algorithm	Description of algorithm
Standard	—	LU decomposition (no enhancements)
T1	Standard	Zero checking
T2	T1	Reordering of circuit nodes
T3	T2	Sparse decomposition of Y matrix
T4	T3	Storage of processed-element array
T5	T3	Sparse matrix storage
T6	T5	Linked-list element storage

4.3 Memory-Dependent Simulator Software

As was mentioned earlier in this section, many memory-speed trade-offs can be made in designing a circuit simulator. The previous section described some speed-dependent aspects of this software. This section covers some memory-dependent aspects of this software.

Two of the largest dimensioned arrays used in circuit simulators are required for the storage of the circuit element data and the admittance matrix entries.

4.3.1 Element Data Storage

The storage of element data in a circuit simulator involves the storage of (1) the element type, (2) its name, value, and circuit node connections, and possibly (3) model information. If a table format

TABLE 9. SUMMARY OF SPEED- AND MEMORY-IMPROVEMENT TECHNIQUES IMPLEMENTED IN BIAS-D

Relative to standard method at 24 circuit nodes (CKT13)

Method	Additional FORTRAN code (lines)	Increase in compiled code (words)	Increase in COMMON arrays (words)	Net memory increase (words)	Relative increase in speed	Relative increase in setup time
Standard	0[a]	0	0	0[b]	1.0	1[c]
T1	10	80	0	80	2.5	1
T2	60	360	0	360	2.8	3
T3	100	580	150	730	5.4	4
T4	110	520	650	1170	5.3	4
T5	170	640	-920	-270	5.9	5
T6	240	880	-2240	-1360	5.7	5

[a] *Standard version of BIAS-D has 1540 lines of FORTRAN, excluding COMMON declarations and Comment statements.*
[b] *Standard version of BIAS-D requires 24,900 decimal words of memory (nonoverlaid) in PRIME 400 minicomputer system; this includes both the FORTRAN and Graphics libraries.*
[c] *For CKT13; setup time is done only once for each circuit; this is a relatively insignificant portion of total analysis time (0.02 s for standard version of BIAS-D).*

is used for this data storage, the maximum circuit size must be predetermined before the program is compiled. This method has one significant advantage; the program is relatively easy to debug or modify. The primary disadvantage of this technique is that the type of circuits analyzed in a general-purpose simulator can vary greatly. Whereas a discrete circuit would have many resistors, capacitors, and possibly inductors, with few transistors, an integrated circuit would have many transistors and capacitors (transistor junction capacitors) with few resistors and no inductors. Thus, in order to handle all circuits, the table storage arrays must be over-dimensioned—wasting memory for the particular circuit at hand. An alternative approach is to use a linked list to store these arrays [6]. Here an entry in each element list points to the next element of that type; the following entries contain the element name, value, etc. This procedure generates a compact single-dimension array in which each particular element type may be linked throughout the list. An additional element type can easily be added to this list with no additional required memory (for list storage), whereas in the table method considerable array space could be required. A more detailed description of how this linked-list array structure is implemented in BIAS-D is given in appendix C.

Both types of these element storage techniques were implemented in BIAS-D. Figure 17 illustrates the memory arrays required to store test circuit CKT13 for both techniques. Figure 17a illustrates the required arrays for the table method, giving both the required size for CKT13 and the maximum dimensioned array size. Figure 17b shows the required array size for CKT13 using the linked-list method. As can be seen in this figure, the table method wastes memory. If a memory comparison for BIAS-D were based on this circuit (CKT13) and both methods were dimensioned (in BIAS-D) such that the minimum dimensions for each method were used, the memory required for the table method would be 1420 words and the memory required for the linked-list method would be 472 words. This is 2/3 less memory.

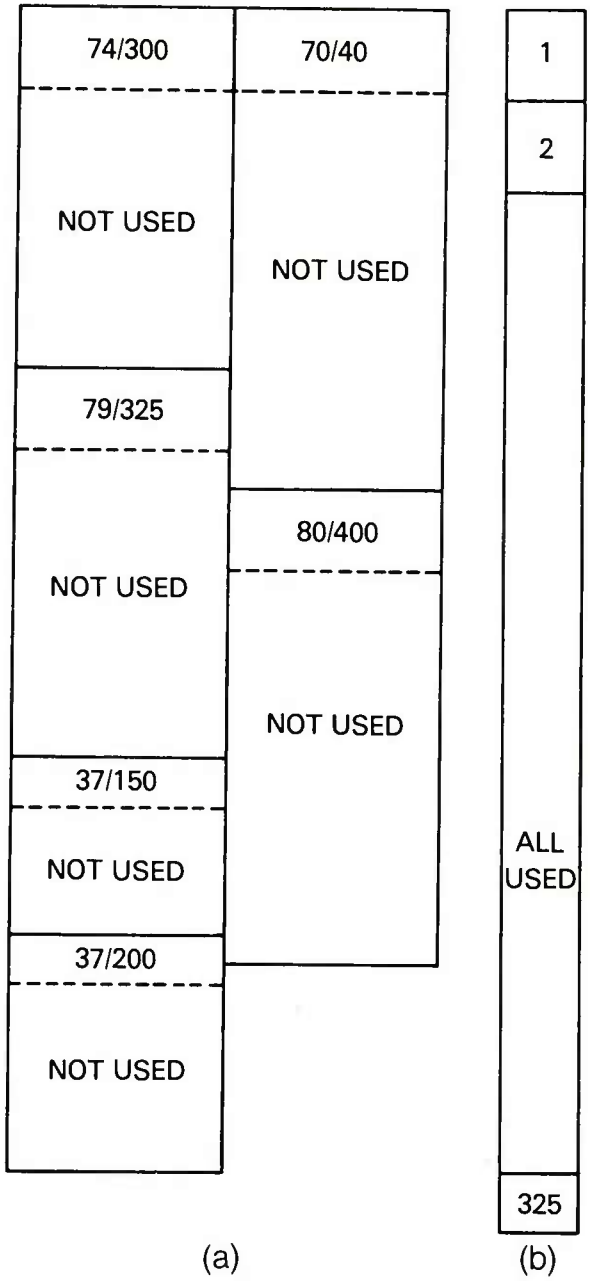


Figure 17. Comparison of element storage requirements in BIAS-D for CKT13 (24 nodes) using (a) table method and (b) linked-list storage.

4.3.2 Sparse Matrix Storage

The pointer structure used to locate the nonzero matrix elements in the admittance matrix can also be used to store only the nonzero elements. This sparse array is then stored as a linear array rather than a matrix array. Additional array space plus an INDEX routine is required to encode the double-dimension address into a linear address.

This sparse storage technique was implemented in BIAS-D (BIAS-T5). Figure 18 graphically compares the two types of matrix storage techniques. Figure 18a shows the memory arrays required using the traditional matrix storage approach used in BIAS-T1 through BIAS-T4 and figure 18b shows that required for the sparse storage approach (in BIAS-T5). The array sizes shown in this figure are for test circuit CKT13. Again, if in each case the arrays for storing the admittance matrix are of minimum dimensions (in BIAS-D), the memory required for the matrix storage of CKT13 is 1936 words and for the sparse storage method (including pointer storage) is 624 words. The sparse storage thus represents a reduction in memory requirements of about 66 percent over the traditional storage (CKT13 is 79-percent sparse).

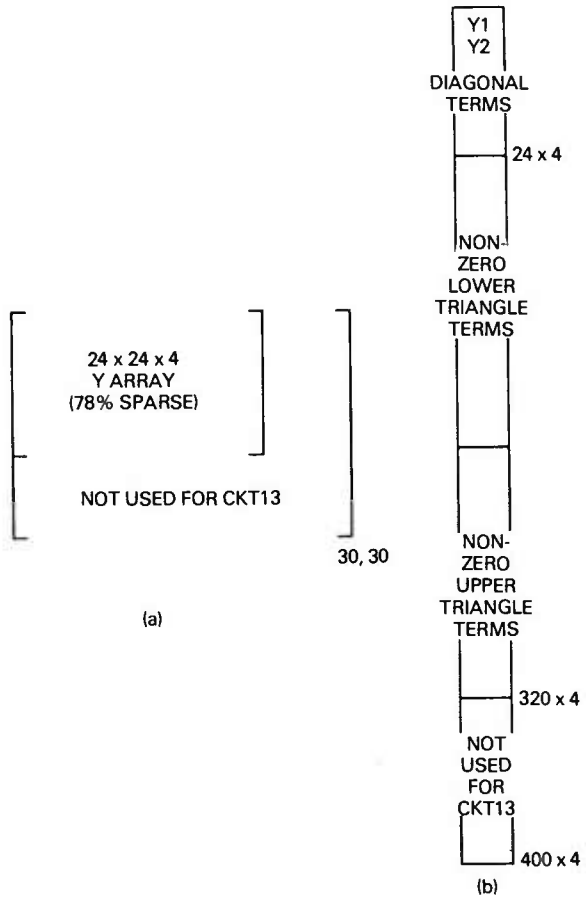


Figure 18. Comparison of storage requirements for admittance matrix of CKT13 using (a) matrix storage and (b) sparse storage.

4.3.3 Memory Overlay

An additional technique for reducing the memory requirements of a program is to use memory overlay [7,30]. Memory overlay requires the use of a disc to store the program segments that are not in use. A single-layer overlay structure for BIAS-D is shown in figure 19. Both common arrays and the main program are resident in main memory at all times. The other overlay segments, the setup overlay, the analysis overlay, and the ac overlay, are in main memory only during execution of that segment. For the example shown in figure 19, the nonoverlaid BIAS-D required 19,600 words of memory (without graphics routines). With the overlay structure shown (fig. 19), the memory requirements are reduced to 14,890 words—a reduction of 4,700 words.

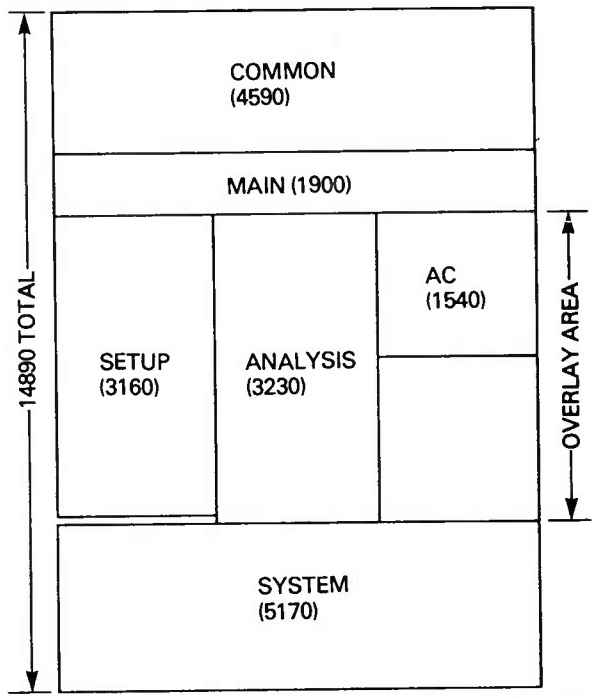


Figure 19. Memory overlay example for BIAS-D.

The primary disadvantage of using memory overlay is that the implementation of these overlays is not compatible between different computer systems. If the memory overlay is not done properly, it can significantly reduce the simulator's execution speed.

4.3.4 *Summary of Memory-Saving Techniques*

These memory techniques, although reducing overall memory requirements, require a certain overhead (both in memory and in analysis speed) to implement. Figure 20 shows a comparison of the analysis speeds for both the linked-list element storage (BIAS-T5) and the sparse storage of the admittance matrix in BIAS-D (BIAS-T6). Also shown in this figure is the speed of BIAS-T3 (sparse decomposition only). Interestingly, the speed improves because of both memory-saving techniques. The linked-list technique was expected to be slower because of the software overhead. Table 9 also gives a summary of the memory-saving techniques with the overhead due to the implementation of these above two techniques in BIAS-D. The table includes (1) savings in dimensioned arrays, (2) increase in memory due to overhead, dimensioned arrays, and software, and (3) speed overhead. As can be seen in table 9, both techniques are worthy of implementation into a circuit simulator because they save memory and increase speed.

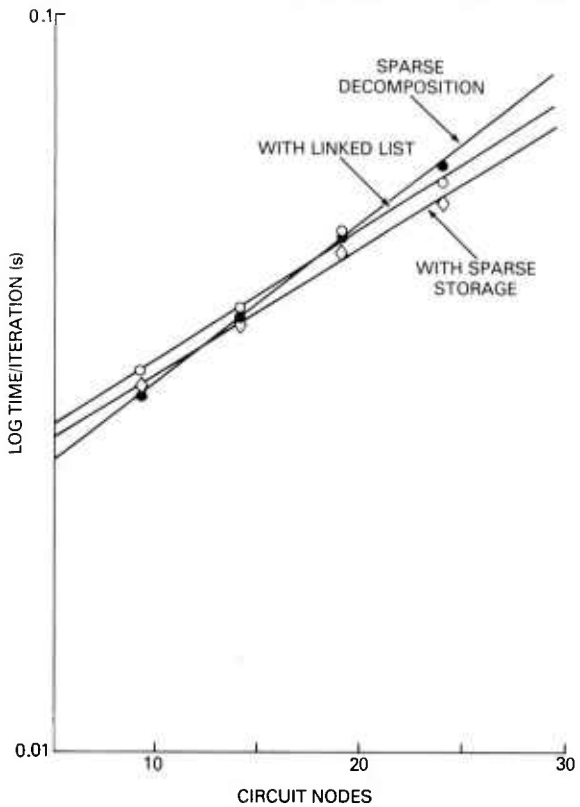


Figure 20. Speed overhead of linked-list and sparse storage techniques.

4.4 Comparison of Minicomputer Systems Using BIAS-D

BIAS-D was again modified to enable comparisons of the minicomputer systems described in section 3. The version used in these comparisons is BIAS-T9 and was obtained by successively modifying BIAS-T8 (see sect. 5) to run on a PDP 11/45, an HP2100 (HP21MX), and the IBM 370/168. BIAS-T8 runs on the PRIME 400 and incorporates all the speed- and memory-saving techniques included in the previous sections, as well as having ac analysis capability. In order to be able to run BIAS-T9 on all these computer systems without source code modifications, it was necessary to delete the graphics capability and other system-dependent routines. Table 10 lists the important details on each of the computer systems used in this comparison. These include the particular operating system in use, the type of memory and its speed, and the version of FORTRAN IV used. Table 11 gives a breakdown of the memory requirements for BIAS-T9 on each of the computer systems. The compiled program size, the size of the common array, and the size of the required system routines (which include the FORTRAN library) are given here. The comparatively large memory requirement of the IBM system is due partially to its use of 4-byte integer words (by default) rather than the 2-byte integer words used by the minicomputer systems. BIAS-D is not dependent on the size of the integer word, and either 2- or 4-byte integers are permissible. The small size of the common array for the HP2100 is due to the 3-word double-precision data word size (see sect. 3). In each computer system, a system-dependent clock routine was added to obtain timing information. The FORTRAN IV source program was input into each computer system via a magnetic tape written in ASCII format (null parity) by the PRIME 400 system.

All benchmark execution-time data were obtained by using BIAS-T9 for the analysis of the

TABLE 10. COMPUTER SYSTEM CONFIGURATION FOR SPEED TESTS

Computer system	Operating system	Double-precision hardware	Cache/ speed	Virtual memory	Memory/ speed
HP2100	RTE II	yes[a]	no	no	Core/1 μs
HP21MX	RTE III	yes	no	no	Core/1 μs
PDP 11/45	RSX 11D	yes	no	no	Core/1 μs
PRIME 400	PRIMOS IV	yes	2k/80 ns	yes	MOS/400 ns
IBM 370/168	MVS2/TSO	yes	32k/80 ns	yes	MOS/320 ns

[a] *Without Fast-FORTRAN read-only memory.*

TABLE 11. COMPARISON OF MEMORY REQUIREMENTS FOR BIAS-T9

Computer system	Common array (bytes)	Compiled program (bytes)	Library and system (bytes)	Total size (bytes)
HP2100	7,810	24,880	—	32,768[a]
HP21MX	7,810	—	—	—
PDP 11/45				
(a) FOR	9,190	32,600	6,160	47,960
(b) F4P	9,190	—	—	49,880
PRIME 400	9,190	19,480	11,230	39,900
IBM 370/168				
(a) opt=0	10,750	44,920	32,430	88,100
(b) opt=3	10,750	33,850	32,450	77,050

[a] *Maximum available memory; without ac analysis.*

TABLE 12. COMPARISON OF COMPUTER SPEEDS FOR CKT13 (BIAS-T9)

Computer system	dc (s)	Transient analysis		Frequency response	
		Time/iter (s)	Speed ratio[a]	Time (s)	Speed ratio[a]
HP2100	21.0/14	682/437	344	—	—
HP21MX	16.4/14	98.2/437	49.6	—	—
PDP 11/45					
(a) FOR	4.75/14	124/441	62.7	41.0	119
(b) F4P	2.10/14	41.3/441	20.9	13.6	25.9
PRIME 400	1.34/14	28.6/441	14.4	10.3	19.6
IBM 370/168					
(a) opt=0	0.125/14	3.10/442	1.56	0.833	1.57
(b) opt=3	0.087/14	1.98/442	1.00	0.525	1.00
SPICE 2C.2 (IBM 370/168 opt=3)	0.191/26	1.42/200	0.717	0.688	1.31
SPICE 2D.2 (PRIME 400)	2.60/13	37.4/218	18.9	—	—

[a] *Relative to IBM 370/168 (opt=3).*

standard test circuits—CKT10 through CKT13. Table 12 gives the execution time for a dc operating point, a 101-point transient analysis, and a 91-point ac analysis (traditional method) for CKT13 for each of the computer systems. The size of the available memory in the HP2100 system (with the RTE II operating system) and the HP21MX (with RTE III) did not permit ac analysis without overlay structures. These routines were therefore deleted from the HP2100/HP21MX version of BIAS-T9. Also given in this table are the speed ratios of each system for the transient and ac analysis for CKT13. Speed comparisons should not be made using execution times for the dc operating points, since for this analysis there is considerable output. In many cases, execution times are somewhat dependent on the terminal's communication rate, which in effect increases execution times.

A comparison of the transient analysis execution times for each of the test circuits CKT10 through CKT13 is given in figure 21. This figure plots log of execution time per transient iteration versus log of the circuit nodes for each of the

computer systems under consideration. Also shown in this figure are the execution times from SPICE1J and SPICE2C.2 on an IBM 370/168 and SPICE2D.2 on a PRIME 400, again using the same test circuits. Data from simulator program Mini-MSINC [2], using the HP2100 minicomputer (with Fast FORTRAN) are also given, and shown as dashed lines in this figure. Although the Mini-MSINC data are from different and unrelated circuits (the MOS model in Mini-MSINC is considerably more complex than the BJT model in BIAS-D), they do indicate the minicomputer's speed.

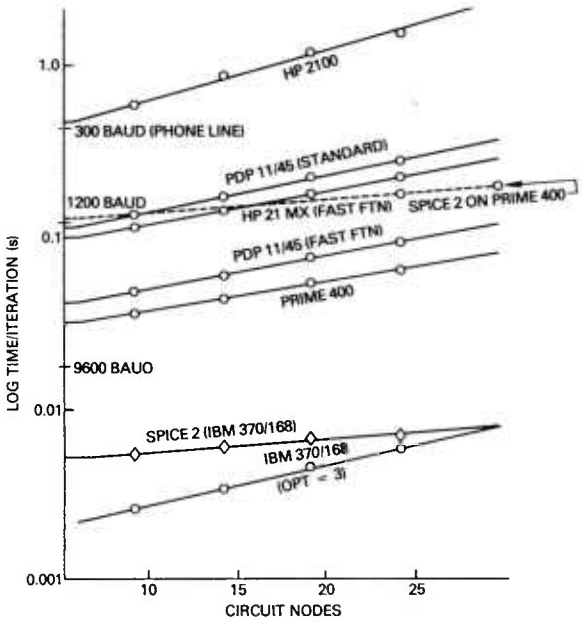


Figure 21. Speed comparison results for different computer systems for transient analysis simulation using four standard test circuits.

A comparison of the execution speeds of BIAS-D and SPICE2 on a PRIME 400 minicomputer (fig. 21, table 11) indicate that BIAS-D is 50 percent faster than SPICE2 on the PRIME 400. This speed difference is due to the large memory requirements of SPICE2. Whereas BIAS-D requires 40 kbytes of memory, SPICE2 requires 400 kbytes. This speed difference demonstrates the advantage gained by developing a circuit simulator specifically for minicomputers.

Also given in figure 21 are the baud rates which could affect output times (wall-clock time but not necessarily CPU time). These points are shown on the vertical axis and were computed based on transmitting a 72-character line with 4.5 iterations between printouts (this was the average iteration count for the transient analysis of the test circuits). If a 1200-baud terminal is used, the results from a transient analysis using BIAS-D on the IBM, PRIME 400, and PDP 11/45 computers appear on the user's terminal at the same wall-clock speed. If a 300-baud terminal is used, results from all computers (with the proper hardware) in this comparison appear at the same rate. These baud-rate limiting points relate only to communication baud rates while BIAS-D is running, but could easily be extrapolated to other simulator programs. The breakpoint in communication speed between using an interactive circuit simulator and a batch simulator is 1200 baud. This choice is based on personal experience and on the experiences of several other users of interactive graphics. The 1200-baud rate is indicated in figure 21. These results indicate that all the minicomputer systems under consideration here are capable of interactive circuit simulation assuming the proper compilers and other hardware are used (see table 10).

5. SMALL-SIGNAL AC FREQUENCY RESPONSE

The ability to compute small-signal frequency response of electronic circuits is important in the design and analysis of linear circuits. There are no general-purpose minicomputer circuit simulators with this capability. Two methods for computing small-signal frequency response have been implemented in BIAS-D—the traditional method, which uses complex matrix operations, and a new method which uses standard transient analysis procedures.

5.1 Traditional Method

The small-signal ac frequency response of an electronic circuit is traditionally found by solving the complex matrix equation [6,39]

$$Y \cdot V = I \quad . \tag{22}$$

The complex admittance matrix, Y, is loaded with the real and imaginary equivalent conductances of

the circuit elements evaluated at the frequency of interest. For active devices, these conductances are determined at the circuit's dc operating points. A complex driving current, given either as the input source or as a Norton equivalent of the driving voltage, is loaded into the complex current vector, I. Equation (22) is then solved for the complex node voltages, V. This method is repeated for each frequency of interest. A flow diagram of the traditional ac analysis procedure is given in figure 22.

If an equivalent ac model for each resistor, inductor, and capacitor for this technique were given, it would be a single complex-value resistor with impedances as shown.

$$\begin{aligned} &\text{Resistor impedance} = R, \\ &\text{Capacitor impedance} = 1/j\omega C, \\ &\text{Inductor impedance} = -j\omega L, \end{aligned} \qquad (23)$$

where ω is the frequency in radians, C is capacitance, and L is inductance.

The primary disadvantage of this technique is that it requires the use of double-precision complex arithmetic. Double-precision complex arithmetic is not available on minicomputer systems and therefore must be added through software (see sect. 3).

5.2 Linearized Transient Analysis (LTA) Method

5.2.1 Large-Signal Transient Response

A new method for determining circuit frequency response without using complex arithmetic, introduced here, uses modified conventional transient analysis techniques. In order to understand this method, the linearized transient analysis method [40], it is important to understand the procedure used to compute large-signal transient response for both linear and nonlinear circuits.

In a large-signal transient analysis simulation, linear capacitors and inductors are modeled by using a conductance in parallel with a voltage-dependent current source as shown in figure 23. The values associated with this model for inductors and capacitors are similar; therefore, for simplicity, only capacitors are considered here. If the trapezoidal integration rule [17] is used to approximate the integral equation for the voltage across a capacitor,

$$V_c = 1/C \int i\,dt = (\Delta t/2C)(i_{t+\Delta t} + i_t) \quad , \qquad (24)$$

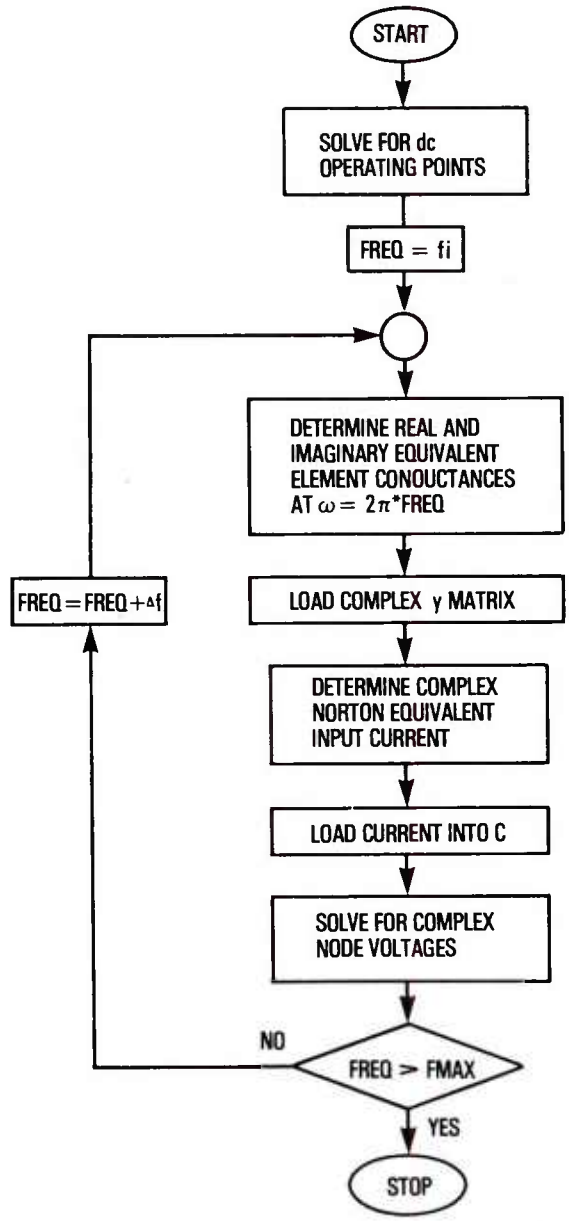


Figure 22. Flow diagram for traditional ac analysis procedure.

where Δ is the time-step, then the equivalent conductance in figure 23 is

$$g_c = 2C/\Delta t \qquad (25)$$

and the equivalent dependent current source for the capacitor model (also in fig. 23) is

$$I_o = I_t + (2C/\Delta t)V_t \quad , \qquad (26)$$

where V_t and I_t are the values of capacitor voltage and current at time t. Currents I_t and I_o are updated and stored at each time-point during a transient analysis. Also at each time-point, the equivalent capacitor conductance, g_c, is loaded into the admittance matrix, Y, and the equivalent capacitor current, I_o, is loaded into the current vector, I. Other circuit element conductances and currents are added into Y and I; and the matrix equation (eq (22) with Y, I, and V real) is solved for the circuit node voltages, V. Equation (22) is solved by using the same procedure as in a dc analysis—that is, LU decomposition, followed by forward and backward substitution [6].

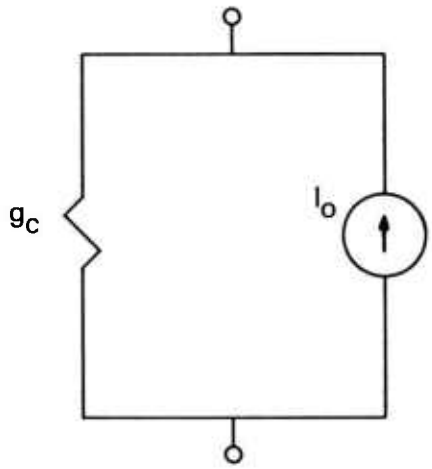


Figure 23. Equivalent circuit for linear time-dependent capacitors or inductors.

If a transient analysis response for a circuit with a sinusoidal input is examined after several periods, the circuit transfer voltage gain and phase shift can be determined. To obtain the overall circuit frequency response, the magnitude gain and phase shift is determined at each frequency of interest. This transient method has several disadvantages, however. (1) For nonlinear circuits it is difficult to choose the input amplitude so that there will be no distortion at the output. (2) The procedure is relatively slow since the admittance matrix must be loaded and the solution must be iterated to convergence for each time-point. (3) It is difficult to determine when the steady-state solution has been reached. For the case of a high-Q circuit, this solution can require many periods [12].

5.2.2 Transient Analysis of Linear Circuits

Looking at the admittance matrix entries for a linear circuit during a transient analysis, one would notice that for a fixed time-step, Δt, all admittance matrix values are constant with time. This constancy means that Y has to be inverted only once for each change in Δt. During a transient analysis with a fixed time-step, Y is inverted only once for a complete transient simulation. The circuit response to any input as a function of time can thus be determined by updating the current vector I, and by doing a simple matrix multiplication (or forward and backward substitution, if LU decomposition is used) at each time-point. If the transient analysis input is sinusoidal then the output is of the form

$$KA \sin(\omega t + \phi) \quad , \qquad (27)$$

where K is the circuit gain at a frequency $\omega/2\pi$, A is the amplitude of the input sinusoid, and ϕ is the output phase shift relative to the input. The amplitude, A, of the input sine wave is not critical in this equation since the circuit is linear. An accurate method of obtaining the output magnitude and phase from this waveform is to use the Fourier approximation for discrete data points [27]. This is essentially a smoothing operation using many data points. Numerical errors that could occur with a single data point are minimized.

Figure 24 is a flow diagram of the procedure to compute the frequency response of a linear circuit at several frequency points, using the transient method just described. Here, rather than the time-step being specified, it is computed from the fre-

quency and the number of computation points per sine wave period (NPTS). During a conventional transient analysis, the procedure at each frequency would include zeroing the capacitor and inductor currents (not shown in fig. 24). This is because each frequency point involves a different sine wave input and is therefore a new analysis.

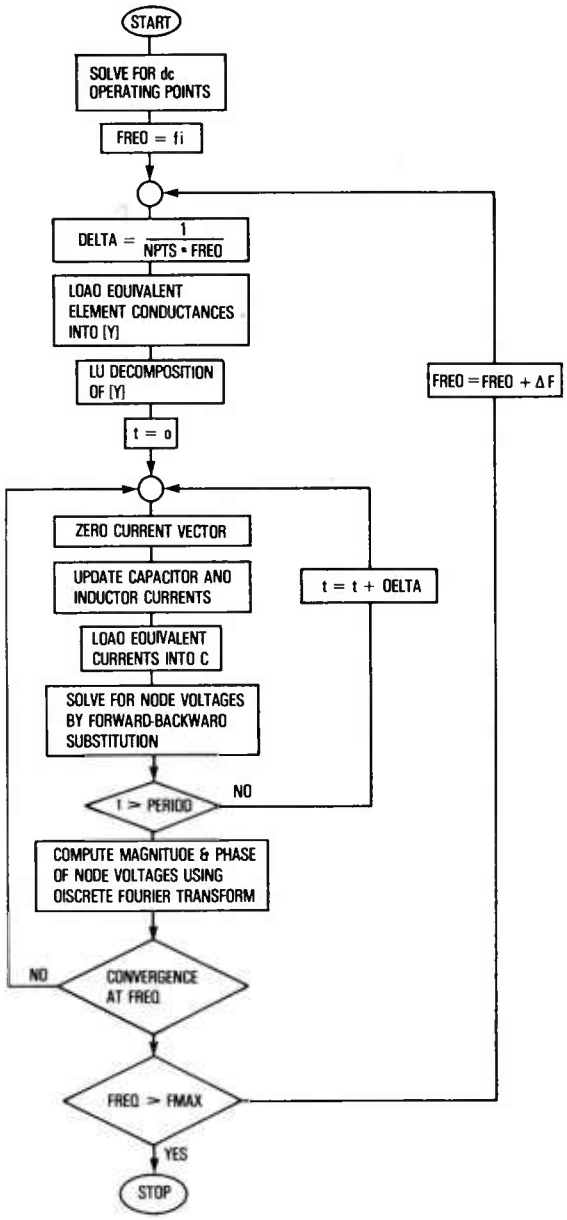


Figure 24. Flow diagram of linearized transient analysis (LTA) method for computing frequency response.

All circuits have an initial transient response to any input or change in input. If there is not an input at t = 0– (dc conditions), then a sine wave applied at t = 0+ would cause a transient response due not only to the sine wave; but also to its application. The effects of this initial transient on the output response of the circuit must decay to zero before these results can be used for determining the frequency response. An advantage of using the Fourier approximation to compute the magnitude of this response is that any dc shift (zero*th* harmonic) due to this transient is separated from the desired output (first harmonic).

The simple bandpass filter circuit shown in figure 25 will be used to illustrate this initial transient response. The voltage transfer function of this circuit is given as

$$V_{out}/V_{in} = As/(1 + Bs + Cs^2) \quad , \qquad (28)$$

where

$$A = R_1C_1 ,$$

$$B = R_1C_1 + R_2C_2 + R_1C_2 , \text{and}$$

$$C = R_1R_2C_1C_2 .$$

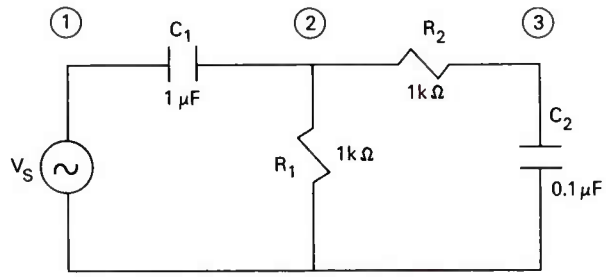


Figure 25. Bandpass filter example circuit.

For $V_{in} = \sin(\omega t)$, the time domain transient response can be determined using the inverse Laplace transform as

$$F(t) = F_1(\omega)\exp(-at) + F_2(\omega)\exp(-bt) + F_3(\omega)\sin(\omega t - \phi) \qquad (29)$$

where

$$\omega = 2\pi f,$$

$$\phi = \text{Arctan}(b/\omega) - \text{Arctan}(\omega/a) + \pi/2,$$

a, b = circuit time constants,
$F_1(\omega)$, $F_2(\omega)$, $F(\omega)$ are dependent on circuit element values and frequency.

The first two terms in equation (28) represent the initial transient response of this circuit. In this case they are both exponential terms decaying in time and inversely proportional to frequency.

5.3 Description of LTA Method

If the transient response procedure shown in figure 24 is begun at frequency f_1, then after several periods at this frequency, the circuit time-dependent currents (I_o in fig. 23) contain the correct steady-state values. These currents are crucial in obtaining the proper circuit magnitude and phase response. At the next frequency point, this procedure is repeated. If this next frequency point is chosen close to f_1, then the values of the time-dependent currents will be approximately those at f_1. If these currents are used as initial conditions for the next frequency point, the number of required periods at each frequency point can be reduced significantly. The use of these currents as initial conditions for the next frequency point is the key to the success of this procedure. Figure 26 shows the transient output response of the circuit in figure 25 to three periods of a 100-Hz sine input followed by three periods of a 110-Hz sine input. In figure 26a, the capacitor current is zeroed at the end of the 100-Hz input. The 110-Hz sinusoid begins as if it were at time t = 0, causing a discontinuity at this point. Figure 26b used the final capacitor current at 100 Hz as the initial current at 110 Hz. Note that although there is a slight discontinuity here, the transition between frequencies is relatively smooth.

A second advantage of retaining the time-dependent currents at the end of each frequency point involves the decay of the initial transient response. Previously, zeroing the time-dependent currents at the end of each frequency point also zeroed time. Thus, the initial transient response began at t = 0+ at each new frequency point. By saving these currents at the end of each frequency, time is continued during the entire analysis (fig. 27). Figure 27 shows only the initial transient response

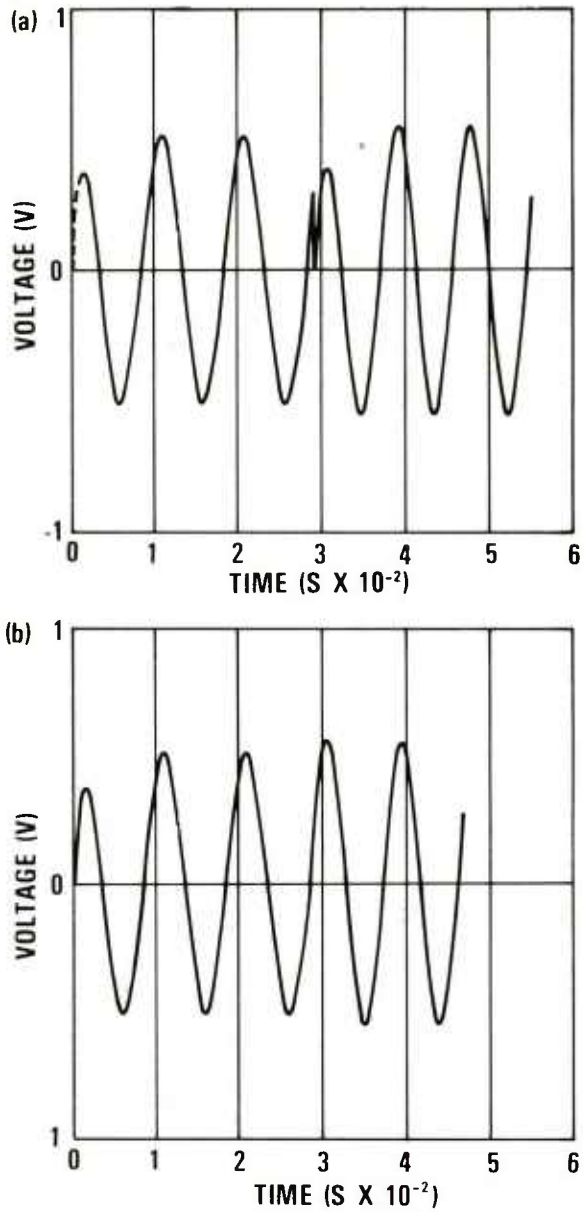


Figure 26. Sinusoidal response of example circuit to three periods of 100 Hz and three periods of 110 Hz: (a) capacitor currents initially zeroed at t = 0 and 110 Hz; (b) capacitor currents zeroed at t = 0 only.

of the results in figure 26. Figure 27a plots the initial transient response with the capacitor current (and time) zeroed at the end of the first three 100-Hz periods. Again note the sharp discontinuity at this point. This is caused by the new zero initial condi-

tions at 110 Hz. Figure 27b shows the same response without zeroing the capacitor currents at the end of the three periods. Here there are no discontinuities. The initial transient decay shown in figure 27 was at a frequency of 100 Hz. If this initial transient is examined at a frequency of 10 Hz, a decade lower, the transient decay is much faster relative to a single 10-Hz period (see eq (29)). This is shown in figure 28. Both the total transient response and the initial transient response are plotted here, using equation (25) (at a frequency of 10 Hz). At this lower frequency, the initial transient response decays about 20 times faster relative to a single sine wave period than at 100 Hz. Beginning at a relatively low frequency can insure decay of the initial transient response. That is, one should choose a starting frequency for the LTA method that is well below the frequency of interest.

5.3.1 Frequency Response of Nonlinear Circuits

The LTA method applied above for linear circuits can be used also with nonlinear circuits. Transistors or other active elements are treated in the same manner as resistors or capacitors; that is, the equivalent ac conductance and currents are added to the admittance matrix and current vector. For transistors, the linearized ac parameters either are already available from the dc operating point calculations or can be computed from the dc operating points. Currents dependent on dc circuit conditions are not added to the current vector. Transistor junction and diffusion capacitor values are determined by the dc operating points and are treated as linear capacitors. As with linear circuits, these equivalent conductances are loaded into the admittance matrix once for each frequency point.

5.3.2 Solution Convergence at Frequency Point

In the LTA method, the sinusoidal input at each frequency point must be continued until the time-dependent currents reach their steady-state values at that frequency. The number of input periods required depends on the frequency increment and the Q of the circuit. A rigorous conver-

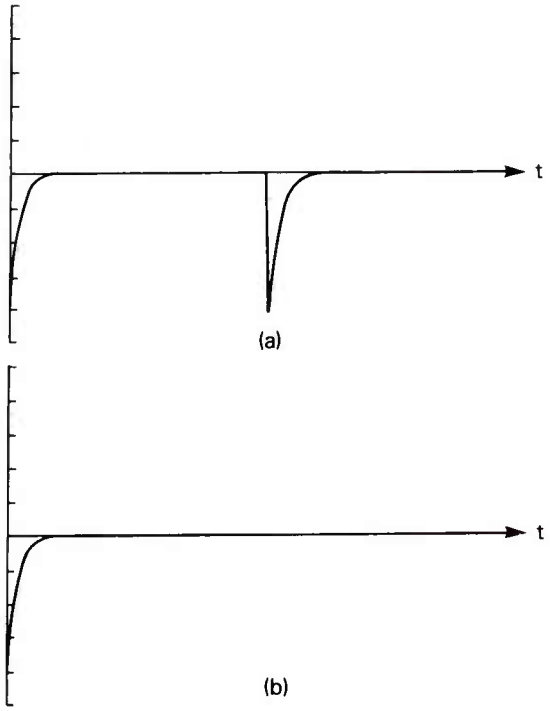


Figure 27. Example circuit sine response for 100 and 110 Hz showing only initial transient: (a) time zeroed at t = 0 and 110 Hz; (b) time zeroed at t = 0 only.

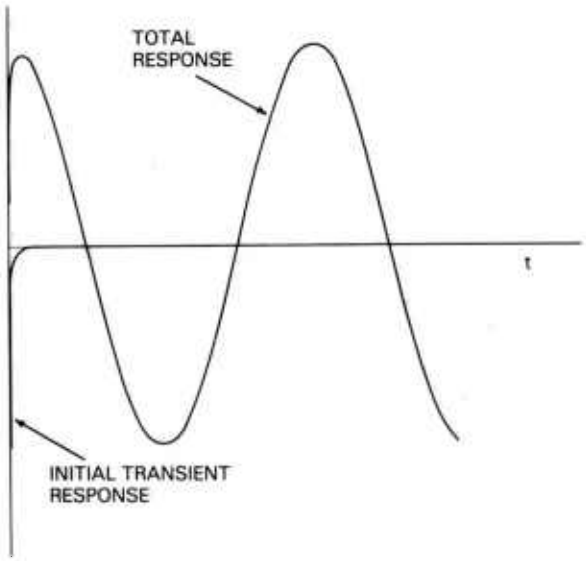


Figure 28. Sinusoidal response of example circuit at 10 Hz showing total response and initial transient response.

gence criterion to determine when a steady state is reached would require all time-dependent currents in the circuit to be within a given tolerance for two or more consecutive periods. This criterion is analogous to the criterion used in the dc and transient analyses. The disadvantage here is that additional storage is required for the past time-dependent currents. Additional computational time also is required to check these currents. A simple technique monitors the change in peak amplitude only at the output node. If the change in amplitude between two periods is less than a predetermined level (say, 1 percent) for two or more consecutive periods, then convergence is assumed. This technique was implemented in BIAS-D and appears to be satisfactory for both high- and low-Q circuits.

5.3.3 Accuracy of Linearized Transient Analysis Method

The accuracy of the LTA method is determined primarily by two factors: the number of analysis points per sine wave period, and the frequency increment between frequency points.

The accuracy of the amplitude is determined primarily by the number of analysis points per period. If the trapezoidal integration rule is used to approximate time-dependent currents, 20 points per period are required to maintain less than a 1-percent numerical integration error in amplitude for low-Q circuits [12]. For high-Q circuits ($Q > 10$), the product of the Q and numerical integration error must be small. It can be shown that the required points per period, n, are proportional to the square root of the Q [41]

$$n = K\sqrt{Q} \tag{30}$$

A conservative value for K is 8 [12].

The accuracy in computing the phase is determined primarily by the frequency increment. If the circuit phase changes rapidly with frequency, then a small frequency increment must be used. The smaller the frequency increment, the smaller the phase change from the previous frequency point.

Choosing a frequency point close to the previous point is essential to the LTA method. If the frequency interval is chosen too large, more periods are required for the time-dependent currents to reach equilibrium. If the interval is too small, excessive points are computed. Both could greatly increase computation time. For circuits with a Q less than one, 10 frequency points per decade appear to be adequate. However, for high-Q circuits, 100 points per decade may be necessary to achieve the desired accuracy. For high-Q circuits, a fixed frequency increment is not desirable, since more points per decade are desired only where the gain changes rapidly with frequency. In this situation a variable frequency-step is best. The frequency can be stepped by a procedure analogous to time-step control used in conventional transient analysis. That is, if more than K periods are required for convergence, then the frequency increment is reduced. If less than M periods are required, then the frequency increment is increased. As in transient analysis, variables K and M are determined by experimentation.

Experimental results using the LTA method have shown magnitude errors less than 5 percent (typically less than 1 percent) and phase errors less than 1 degree (typically 0.2 degrees). These errors were obtained from several low-Q circuits by using 10 computation points per period and 20 frequency points per decade (see sect. 5.3.4 for examples showing these errors).

5.3.4 Comparison with Traditional ac Method

The LTA method was implemented in circuit-simulator program BIAS-D as BIAS-T7. A flow diagram of the LTA procedure is shown in figure 24. The number of points per period, the number of points per frequency decade, and the maximum number of periods were input variables. The convergence criterion required the amplitude difference among three consecutive periods to be less than 1 percent. This criterion does not imply that the magnitude error will be less than 1 percent.

Since dc and transient analysis capability were already implemented in BIAS-D, only one addi-

tional FORTRAN subroutine was required to implement this method. No additional dimensioned arrays were required.

The traditional method using complex matrix inversion also was implemented in BIAS-D (BIAS-T8). This method was used to compare speed of the memory and accuracy with those of the LTA method. Seven additional subroutines were required to implement this traditional method. Most of these subroutines were identical to existing routines in BIAS-D but included complex arithmetic operations. Since minicomputers do not support double-precision complex arithmetic, these operations were programmed into the software (see sect. 3). The storage of the complex matrix equation—equation (22)—required doubling the size of the original Y, V, and I arrays. These are double-precision arrays and require significant additional memory.

With both the traditional and LTA methods implemented in BIAS-D it was possible to compare the accuracy of the LTA method with that of the traditional. The first comparison was made by using test circuit CKT10 (app C). Figure 29a plots the decibel gain of this circuit versus frequency for both the LTA method and the traditional method (the LTA method is plotted as solid lines). Figure 29b plots phase for both methods. Ten frequency points per decade were used in both cases. For the LTA method, 20 points per period were used. Excellent agreement in both gain and phase was obtained (less than 1-percent magnitude and 0.25-degree phase differences). A total of 336 periods was required for the 91 frequency points in the LTA method. This is an average of 3.7 periods per frequency point (3 periods is the minimum allowed). Further, 32 s of CPU time was required (on a PRIME 400 minicomputer) for the LTA method, whereas 4.4 s were required for the traditional method (a 7:1 speed ratio).

The simple tuned circuit (with a Q of 25) [12] shown in figure 30 was used to compare results for high-Q circuits. The magnitude and phase of this circuit to a current input for both methods are given in figure 31 (the LTA method in solid lines). One

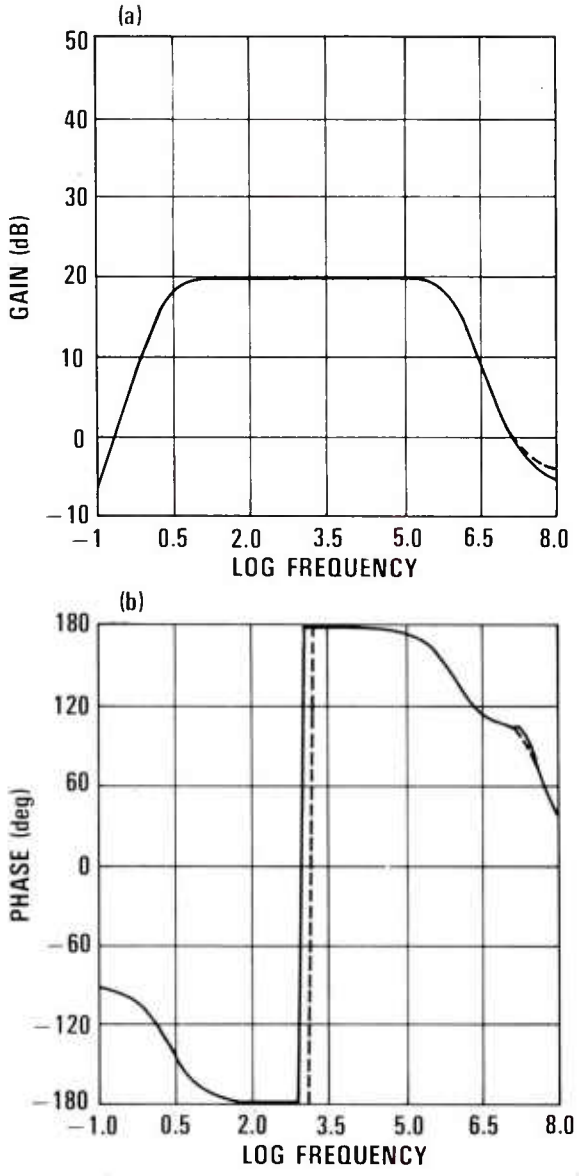


Figure 29. Frequency response of test circuit CKT10 comparing both methods computed using 10 points/decade and 20 points/period: (a) magnitude and (b) phase.

hundred points per decade and 40 points per period (as required by eq (26)) were used in the analysis. Again excellent agreement was obtained. A total of 1492 periods was required in the LTA method for 100 frequency points (an average of 15 periods per frequency point). The CPU time for the

LTA method was 88.9 s. The time for the standard method was 0.891 s (a 100:1 speed ratio). Figure 32 shows magnitude and phase results from the same high-Q circuit with the frequency points per decade for both methods decreased to 20 and the points per period for the LTA method decreased to 20. As expected, errors in both magnitude and phase have increased. The choppiness in this plot is due to the small number of frequency points plotted (20 points). These results required 476 periods for the 20 frequency points or an average of 21 periods per frequency point. This increase in periods was expected because of the larger frequency increment. The CPU time for the LTA method was 13.1 s, whereas the traditional method required 0.194 s (a 68:1 speed ratio).

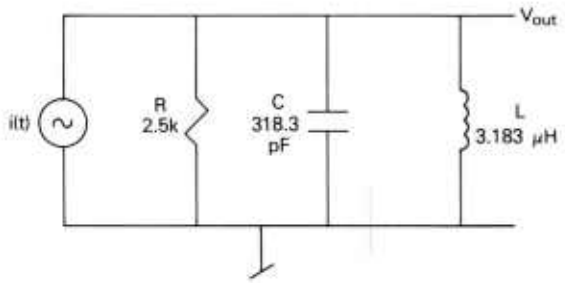


Figure 30. High-Q example circuit.

An analysis speed comparison of these two ac analysis techniques was made using BIAS-T7 and BIAS-T8. Test circuits CKT10 to CKT13 were used on the PRIME 400. These were the same circuits used in section 4. Figure 33 shows the results of these tests. Circuit nodes are plotted versus log-CPU time. Also shown in this figure is the CPU execution speed of these circuits using SPICE2 on the PRIME. As can be seen in this figure, the LTA method is significantly slower than the traditional method. The fact that SPICE2 runs more slowly than BIAS-T8 on the PRIME 400 is due to the large memory requirements for this system. Comparisons in section 4 (table 12) indicate that the ac analysis speed on BIAS-T8 (BIAS-T9) is approximately the same as SPICE2.

A summary of the required memory for implementing both the traditional and LTA methods in BIAS-D is given in table 13.

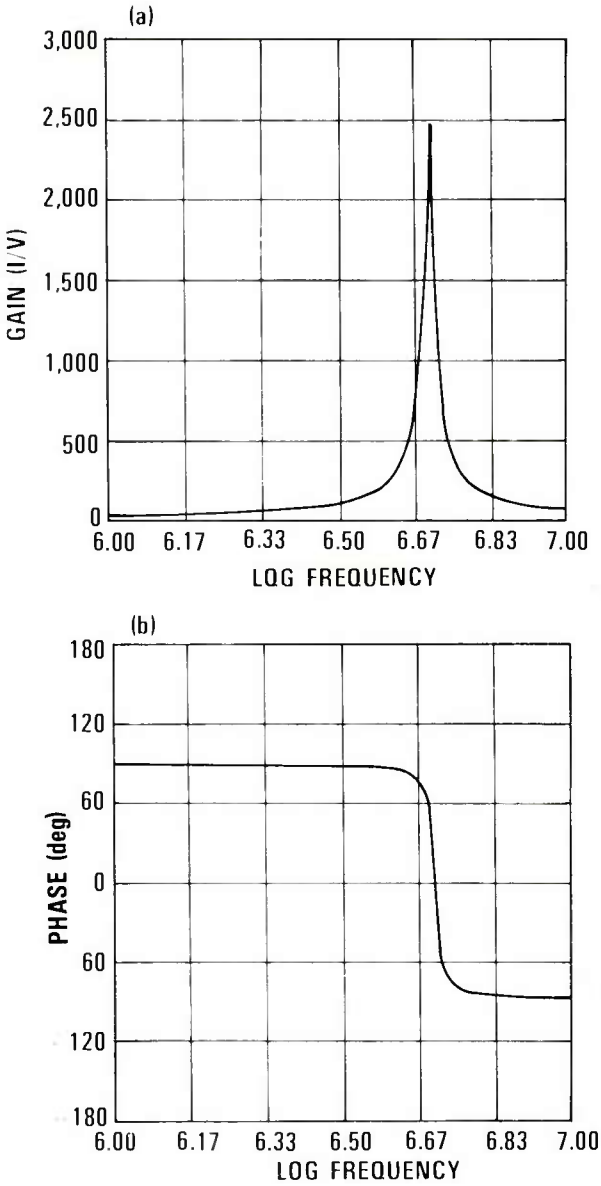


Figure 31. Frequency response of high-Q example circuit comparing both methods, using 100 points/decade and 40 points/period: (a) magnitude and (b) phase.

As can be determined from this table, at the 30-node level, the LTA method does not represent a significant memory saving (18 percent). However, at the 100-node level the memory savings increase to 50 percent (13,000 words) and at the 1000-node level to 95 percent (250,000 words).

Although the LTA method is slower than the traditional method, the memory savings can be significant. The savings increase rapidly with increasing circuit size. Whereas the traditional method requires additional memory to store the complex admittance matrix, the LTA method needs no additional memory for frequency response analysis at any circuit size. The speed of the LTA method

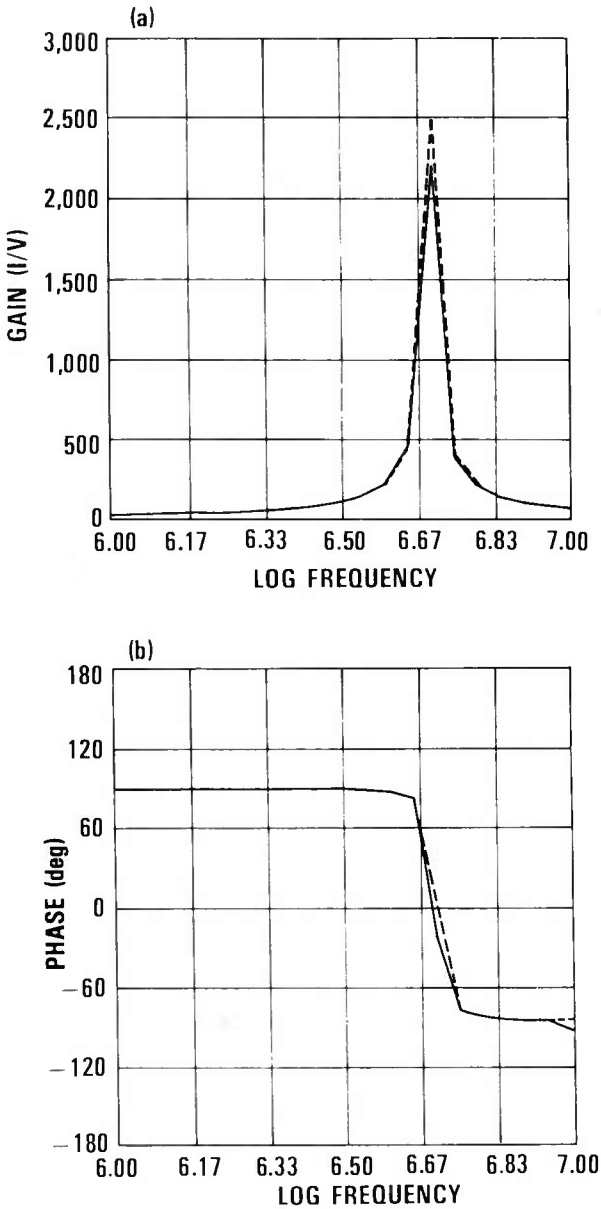


Figure 32. Frequency response of high-Q example circuit comparing both methods, using 20 points/decade and 20 points/period: (a) magnitude and (b) phase.

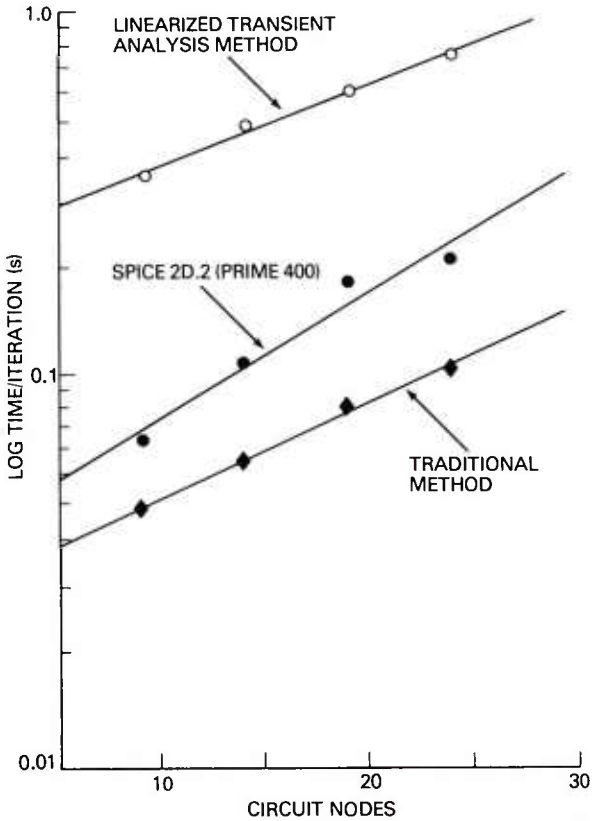


Figure 33. Comparison of frequency response speeds using traditional method and LTA method in BIAS-D and traditional method in SPICE 2D (all on PRIME 400).

TABLE 13. SUMMARY OF MEMORY NEEDS FOR TWO METHODS

Method	FORTRAN (lines)	Common (words)	Compiled program (words)	Total[a] (words)
Traditional	400	1840	2100	3940
Linearized transient analysis	140	0	950	950

[a] *The total program size for BIAS-D (30 nodes) without ac is 12,800 words.*

could be increased by possibly 25 percent at the expense of using additional memory. However, the primary reason for the development of this method was to minimize memory requirements.

Because of its slow speed, practical use of the LTA method for frequency response analysis is limited to the desktop calculators and small minicomputer systems where memory is limited and complex arithmetic is not easily attained. Other application areas should be investigated, such as determining the steady-state response of lightly damped circuits or large-signal harmonic distortion analysis.

6. CONCLUSIONS

Developments in the sections 2 to 5 determined that circuit simulation on small computer systems is both practical and desirable. Both interactive and batch simulator architectures were described, with the interactive simulator being the most desirable.

The second section concerned circuit simulation on programmable desktop calculators. BIAS-D has shown that interactive circuit simulation on desktop calculators is indeed possible; however, as in the case of the speed limitations of the HP9830A, it is not practical. BIASL.25 on the HP9825 offers a significant speed increase (approximately 10:1) but the use of HPL limits its use to the HP9825. Recently available second-generation "super calculators" such as the HP9845 or the Wang PCS-II are as fast or faster than the HP9825 and use the BASIC language. These calculators should make interactive simulation at the 10- to 20- node level practical.

The third section introduced facets of small computer systems useful in development of circuit-simulator programs—the computer language, data word format, computer and language speeds, and memory configuration.

In order for a simulator to be easily transportable between computer systems, FORTRAN IV should be used wherever possible. In the case of the desktop calculator, BASIC should be used, since FORTRAN is not yet available. The use of virtual memory in computer systems offers a significant advantage in increasing software transportability—especially for large programs.

The data word format used in minicomputers for alphanumeric, integer, and single-precision variables should present minimal problems when used in minicomputer simulators. Simulators which are to be transportable between several minicomputer systems should use word-oriented alphanumeric variables (i.e., one ASCII character per word) rather than byte-oriented variables. Double-precision data word formats varied considerably—from 10 digits for the HP2100 to 27 digits for the CDC 6600. The 10-digit precision of the HP2100 double-precision arithmetic could present difficulties in some simulator algorithms. In this case the work of Freret [9,27,28] on word-length limitations should be considered.

In section 4, on circuit simulation on minicomputers, we determined that, for effective interactive simulation, a relatively fast minicomputer should be used. Slow computers result in excessive wait times for the interactive process; for such systems, batch simulators should be used. Experimental results from BIAS-D show that the techniques used in larger simulator programs, such as SPICE, can also be used efficiently in minicomputer simulators with no loss in accuracy. The following techniques can be used in minicomputer circuit simulators to minimize memory requirements and maximize speed:

1. sparse matrix decomposition with node reordering,
2. sparse matrix storage, and
3. linked-list element storage.

A surprising result was that the storage of element "templates" [2,6] or locations for adding element equivalent conductance values to the admittance matrix used considerable memory with little improvement in speed (up to the 50-node circuit level). Speed comparisons of the HP2100, the PDP 11/45, the PRIME 400, and the IBM 370/168 computer systems indicate that with the proper software and hardware configurations, all are capa-

ble of effective interactive circuit simulation at the 30- to 50-node level.

Section 5 introduced the linearized transient analysis method for computing small-signal frequency response, a technique using no complex arithmetic and significantly less memory than the conventional method. Because of its speed handicap, the practical use of the LTA method for small-signal frequency response is limited to smaller minicomputers and desktop calculators where memory is limited. Other application areas for this technique, such as determining steady-state transient operating points, should be investigated further.

LITERATURE CITED

1. B. L. Biehl, BIAS-D: A Semi-Interactive Circuit Analysis Program for Desktop Calculators and Minicomputers, Eighth Annual Asilomar Conference on Circuits, Systems and Computers, December 1-3, 1974.

2. T. K. Young and R. W. Dutton, Mini-MSINC—A Minicomputer Simulator for MOS Circuits with Modular Built-in Models, IEEE J. Solid-State Circuits, *SC-11*, No. 5, 730-732, October 1976.

3. A. R. Newton and G. L. Taylor, BIASL.25, A MOS Circuit Simulator, Tenth Annual Asilomar Conference on Circuits, Systems and Computers, November 22-24, 1976.

4. T. E. Idleman, F. S. Jenkins, W. J. McCalla, and D. O. Pederson, SLIC—A Simulator for Linear Integrated Circuits, IEEE J. Solid-State Circuits, *SC-6*, 188-204, August 1971.

5. ASTAP General Information Manual (GH20-1271-0), International Business Corp., Mechanicsburg, PA.

6. L. W. Nagel, SPICE 2: A Computer Program to Simulate Semiconductor Circuits, Electronics Research Laboratory, ERL-M520, University of California, Berkeley, May 1975.

7. E. Cohen, Program Reference for SPICE 2, Electronics Research Laboratory, ERL-M592, University of California, Berkeley, June 1976.

8. A. R. Newton and D. O. Pederson, The State of Integrated Circuit Simulators, Proc. Midwest Cir. Theory Symp., August 1977.

9. J. P. Freret, Jr., Overcoming Wordlength Limitations in Minicomputer Aided Circuit Analysis, Ph.D. Dissertation, Stanford University, Stanford, California, May 1976.

10. B. L. Biehl, Circuit Simulation on Minicomputers, Asilomar Conference Digest, November 1976.

11. W. J. McCalla and W. G. Howard, Jr., BIAS-3—A Program for the Nonlinear dc Analysis

of Bipolar Transistor Circuits, IEEE J. Solid-State Circuits, *SC-6*, 14-19, February 1971.

12. S. P. Fan, Sinc-S: A Computer Program for the Steady-State Analysis of Transistor Oscillators, Ph.D. Dissertation, University of California, Berkeley, September, 1975.

13. J. J. Ebers and J. L. Moll, Large Signal Behavior of Junction Transistors, Proc. IRE, *42*, 1761-1772, December 1954.

14. H. C. Lin, Integrated Electronics, San Francisco, CA, Holden-Day, 1967.

15. J. M. Early, Effects of Space-Charge Layer Widening in Junction Transistors, Proc. IRE, *46*, 1141-1152, November 1952.

16. R. Barham, E. Cheung, and E. Cohen, BIAS-M, An Experimental Circuit Simulator for the IBM 1800, Integrated Circuits Group, University of California, Berkeley, June 1973.

17. H. W. Dommel, Digital Computer Solution of Electromagnetic Transients in Single and Multiple Networks, IEEE Trans. Power Appar. Syst., *PAS-88*, 378-385, August 1970.

18. L. O. Chua and P. M. Lin, Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques, Englewood Cliffs, New Jersey, Prentice-Hall, 1975.

19. T. K. Young and R. W. Dutton, Local Truncation Error Control for Circuit Simulators, Proc. Midwest Cir. Theory Symp., August 1977.

20. M. J. Hellstrom et al, An Integrated Circuit Preamplifier with Nonlinear Bootstrapped Input Impedance, Proc. Natl. Electronics Conf., *23*, 321-324, 1967.

21. G. J. Vosatka, The Minicomputer—Evolution or Revolution, Minicomputer Trends and Applications 1973 Symposium Record, April 4, 1973.

22. Hewlett Packard FORTRAN IV Reference Manual, Hewlett Packard Corp.

23. PDP-11 FORTRAN Language Reference Manual Dec-11-LFLRA-B-D, Digital Equipment Corp., Maynard, MA, 1974.

24. PRIME FORTRAN IV Users Guide Revision D MAN1674, PRIME Computer Corp., June 1976.

25. Control Data 6400/6500/6600 Computer Systems FORTRAN Reference Manual, Control Data Corp., 1969.

26. IBM System/370 Principles of Operation GA 22-7000-5, International Business Machines Corp., 1974.

27. A. Ralston, A First Course in Numerical Analysis. New York, McGraw-Hill, 1965.

28. J. P. Freret, Minicomputer Calculation of the Dc Operating Point of Bipolar Circuits, Technical Report No. 5015-1, Stanford University, Stanford, CA, May 1976.

29. J. P. Freret and R. W. Dutton, Successful Circuit Simulation using Minicomputers, 19th Midwest Symposium for Circuits and Systems, Milwaukee, WI, August 1976.

30. H. S. Stone, Introduction to Computer Architecture, Chicago, Illinois: Science Research Associates, Inc., 1975.

31. Wang 2200A/B BASIC Programming Manual, Wang Laboratories Inc., 1974.

32. Hewlett Packard 9830A Calculator Operating and Programming Manual, Hewlett Packard Calculator Products Division, 1973.

33. Tektronix 4051 Graphic Systems Reference Manual, Tektronix Inc., 1976.

34. F. S. Jenkins and S. P. Fan, Time—A Nonlinear DC and Time-Domain Circuit Simulation Program, IEEE J. Solid-State Circuits, *SC-6*, 182-188, August 1971.

35. T. K. Young and R. W. Dutton, MSINC, Stanford Electronics Laboratories, SU SEL-74-038, TR 501U, July 1974.

36. ISPICE Reference Guide, form 968-2, National CSS, Inc., Norwalk, CT, 1974.

37. H. M. Markowitz, The Elimination Form of the Inverse and its Application to Linear Programming, Management Sci., *3*, 255-269, April 1957.

38. A. R. Newton and D. O. Pederson, Analysis Time, Accuracy, and Memory Requirement Tradeoffs in SPICE2, Eleventh Annual Asilomar Conference on Circuits, Systems and Computers, November 1977.

39. D. A. Calahan, Computer-Aided Network Design, Revised Edition, New York, McGraw-Hill, 1972.

40. B. L. Biehl, A Linearized Transient Analysis Technique for Computing Frequency Response in Circuit Simulators, Eleventh Annual Asilomar Conference on Circuits, Systems and Computers, November 1977.

41. T. J. Aprille, Jr., and T. N. Trick, Steady-State Analysis of Nonlinear Circuits with Periodic Inputs, Proc. IEEE, *60*, 108-114, January 1972.

APPENDIX A. BIAS-D USER'S MANUAL (BASIC VERSION) AND LISTING

A-1. INTRODUCTION

BIAS-D is a computer-aided circuit-analysis program written in BASIC for desktop calculators and minicomputers with a minimum of 8 kwords of internal memory. It can perform dc and transient analysis of a 15-node circuit that contains up to 75 elements—resistors, capacitors, voltage sources, current sources, and transistors (15 each). For transistor circuits, BIAS-D converges to a solution by linearizing the built-in Ebers-Moll transistor model about an operating point in much the same manner as done in larger circuit-analysis programs such as BIAS-3, SLIC, and SPICE.

Circuit data are typed into the keyboard in a semifree input format. Error messages are given for recoverable data errors enabling immediate corrections. Transistor parameters, temperature coefficients, and transient sources are entered by specifying one or more of five available model types.

BIAS-D executes in a semi-interactive mode in which elements or models are altered, temperature varied, and elements inserted between existing nonsource nodes. BIAS-D is structured so that the circuit size and element capacity can be easily modified in accordance with the available memory size. Execution time for a dc solution of a 10-node, 5-transistor circuit is approximately three minutes on an HP9830A desktop calculator.

A-2. INPUT DATA

The input data are divided into two categories: circuit data and control statement data. The circuit element data (e.g., resistors, transistors, etc) are input by specifying the element symbol (R, Q, M, etc) followed by the required data for that element. The control statement data are characterized by a dot (.) followed by the desired operation (.TRAN, .ALTER, etc). Control statements do not affect the results of the analyses—they only enable the user to direct the analysis procedure.

A-2.1 Circuit Data

Certain general instructions must be followed to input circuit data.

a. Each circuit element must begin in column 1.
b. Single spaces are used as delimiters between data fields (multiple spacing may result in errors).
c. Abbreviated notation cannot be used (i.e., 2U#2E–6).
d. Scientific notation may be used (i.e., 1000 = 1E3).
e. Decimal points are not required (i.e., 2 = 2.0).
f. The ground node must be node 0 (zero).
g. Compact node numbering is not required (i.e., node numbers may be skipped).
h. The maximum allowable node number is 99.
i. Element values are to be in basic units (i.e., ohms, farads, volts, amperes, hertz, seconds).

A-2.1.1 Resistors, Capacitors

General form:

```
RX N1 N2 VALUE
CX N1 N2 VALUE
```

where X is any character, N1 and N2 are node numbers (order not important), and VALUE is the resistor or capacitor value in ohms or farads. Note: VALUE cannot be zero.

A-2.1.2 Independent Sources: Voltage, Current

General form:

```
VX N+ N– VALUE M#
IX N+ N– VALUE M#
```

where X is any character, N+ and N– are the positive and negative source nodes, and VALUE is the source value in volts or amperes. The letter M followed by an integer from 1 to 5 denotes the model name (see sect. A-2.1.4).

For voltage sources, either N+ or N– must be grounded (node 0). For example,

V+ 3 0 5 M1

and

V+ 0 3 –5 M1

are equivalent.

For current sources, current flows from the positive node through the source to the negative node. The letter M followed by the model name may be omitted. However, a default number of zero is assigned.

A-2.1.3 Transistors

General form:

QX NC NB NE M#

where X is any character, and NC, NB, and NE are the collector, base, and emitter node numbers, respectively. The letter M followed by an integer from 1 to 5 denotes the model name (see sect. A-2.1.4). The letter M followed by the model name may be omitted. However, a default number of zero is assigned.

A-2.1.4 Model

General form:

M# YYY F1 F2 F3 F4 F5 F6

where # is an integer from 1 to 5 corresponding to the model number designated by the source or element. YYY is a three-letter name designating one of five available model types as follows:

1. NPN npn transistor parameters
2. PNP pnp transistor parameters
3. PUL pulse source specifications
4. SIN sinusoidal source specifications
5. TEM element temperature coefficients

F1, F2, . . . , F6 are the data fields for specifying the above model parameters. These fields are defined below.

1. NPN—transistor parameters

Field	Parameter	Default value
F1	Forward dc beta (B_F)	100
F2	Reverse dc beta (B_R)	1
F3	Saturation current (I_S)	1E–15
F4	Early voltage (V_A)	1E+12
F5	Recombination current parameter (collector current at which beta = $B_F/2$)	0
F6	Not used	—

2. PNP—transistor parameters (same as NPN)

3. PUL—pulse source specifications

Field	Parameter	Default value
F1	Initial source value at t = 0	0
F2	Pulsed value	0
F3	Pulse delay time	0
F4	Pulse rise time	0
F5	Pulse duration (width)	0
F6	Pulse fall time	0

4. SIN—sinusoidal source specification

Field	Parameter	Default value
F1	dc source value (offset)	0
F2	Source amplitude (0-P)	0
F3	Source frequency (Hz)	0
F4	Time delay	T_{step}
F5	Phase shift (deg)	0
F6	Not used	—

The value of the sinusoidal source is determined by the equation

$$F(t) = F1 + F2 \sin [2\pi F3(t - F4) + F5]$$

5. TEM—element temperature coefficients

Field	Temperature coefficient	Default value
F1	Resistor (T_{C1})	0
F2	Resistor (T_{C2})	0
F3	Capacitor (T_{C1})	0
F4	Capacitor (T_{C2})	0
F5	Transistor beta (T_{C1})	0
F6	Transistor beta (T_{C2})	0

The element value at temperature T is determined by the equation

$$E(T) = E(T_o)[1 + (T - T_o)T_{C1} + (T - T_o)^2 T_{C2}]$$

where $T_o = 300$ K. T_{C1} and T_{C2} are the element's first- and second-order temperature coefficients, respectively. The dimensions of T_{C1} and T_{C2} are in decimal percentages per degree Celsius (a decimal percent of 0.002 = 2000 ppm/C).

A-2.1.5 Comment Statement

General form:

* any comment

A comment may be inserted at any line in the input circuit by using an asterisk (*) in column 1 followed by any message up to 80 characters long.

A-2.1.6 END Statement

END terminates the inputting of circuit data. If a default transistor model is used, it may be necessary to use END twice in succession. (Note: on the HP9830 this is not the same as the END key.)

A-2.2 Control Commands

After each type of analysis is completed, program control is returned to the operator. This is indicated by "INPUT CARD" appearing on the display. At this time it is possible to initiate a new analysis. This is done by using one of the control commands described in the following sections; all control commands are prefixed by a dot(.).

A-2.2.1 .ALTER

The .ALTER command enables element values, models, and model parameters to be altered. This is done as follows.

```
.ALTER
RX  VALUE
VX  VALUE
      .
      .
      .
END
```

where X is a valid element name (i.e., has been previously defined) and Value is the new element value. One or more element values may be altered using a single .ALTER command. An END statement terminates the alter operation. Models and model parameters can be altered in the same manner as the elements. Model types may be changed by entering a different three-letter designation (see sect. A-2.1.4). For example, a pulse source PUL can be changed to a sinusoidal source, SIN, etc. All model parameters must be entered or they are set to their default values. Both models and elements can be altered at the same time.

A-2.2.2 .INSERT

The .INSERT command permits elements or models to be inserted into an existing circuit. The use of this command is limited to insertion of elements and current sources between existing nodes which are not connected to a voltage source (except node 0). Any type of model may be inserted. The .INSERT command is used as follows.

```
.INSERT
RX  N1  N2  VALUE
QX  NC  NB  NE  M#
M#  YYY  F1  F2  F3  F4  F5  F6
      .
      .
      .
END
```

The format for the elements and models is the same as described at the beginning of section A-2.

A-2.2.3 .GAIN

The small-signal ac gain and input impedance between any two nodes (and ground) can be determined using the .GAIN command. This is done as follows.

```
.GAIN
"INPUT NODE"
(enter input node)
"OUTPUT NODE"
(enter output node)
```

Gain and input impedance are printed out. This procedure is repeated for each new gain calculation. (Note: gain cannot be computed at a source node.)

A-2.2.4 .TEMP

The analysis of the circuit at a temperature other than 27 C is obtained as follows.

```
.TEMP
"TEMPERATURE(DEG C)?"
(enter temperature)
```

This procedure is repeated for each new temperature. If a TEM model has not been defined, "ILLEGAL CHARACTER" will be displayed. This model can be inserted using the .INSERT command. (Note: any subsequent analysis is performed at the last temperature specified.)

A-2.2.5 .TRAN

A transient analysis can be obtained using the .TRAN command as follows:

```
.TRAN
"TIMESTEP=?"
(enter time-step)
```

In order for the transient analysis to be meaningful, one or more source models (SIN, PUL) must have been specified. Voltage sources cannot be added once the initial circuit has been entered; however, source models can be inserted or altered, except for M0 (see also sect. 2.2.6). A dc transfer curve can be obtained using the .TRAN command. This is done using the PUL model with such parameters that the pulse rise-time is long compared to the circuit time constants.

A-2.2.6 .OUTPUT

The output voltages of up to five nodes may be simultaneously printed for each timepoint in a transient analysis. This is done using the .OUTPUT command as follows:

```
.OUTPUT
"OUTPUT NODE?"
(type desired output node)
```

This procedure is repeated for each output node (to a maximum of five outputs).

A-3. MISCELLANEOUS (HP9830A)

A-3.1 Early Termination

In some cases it may be necessary to terminate an analysis before completion. This can be accomplished using the END key if the program has stopped or the STOP key if the program is running. This terminates program control. Variable values can be examined at this time. Program control can be regained by one of the following sequences.

1. CONTINUE EXECUTE: This continues the program at the point the END key was depressed.

2. CONTINUE 140 EXECUTE: The old analysis is terminated (the circuit is still retained, however) and the program waits for a new control command (i.e., .ALTER, .TRAN, etc).

A-3.2 Mean Error Printout

Sometimes convergence to the desired accuracy is not attained. If this happens, a "MEAN ERROR:" printout will occur. These results may or

may not be correct. If, during a dc analysis, a more accurate solution is desired, the following procedure can be used.

```
.ALTER
END
```

This does not change the circuit but allows at least four more iterations to occur.

A-4. BIAS-D SOURCE LISTING (BASIC)

A listing of the BASIC version of BIAS-D is given here. This listing is directly compatible with an HP9830A desktop calculator with a string-variable ROM and a matrix operations ROM. Minor modifications are required for execution on a Wang 2200 or a Tektronix 4051.

```
10   REM   ***************** BIAS-D *****************
20   REM   CIRCUIT ANALYSIS PROGRAM-VERSION 2 MOD 8 11-14-74
30   REM    B.BIEHL, HARRY DIAMOND LABS WASHINGTON DC
35   OPTION BASE 1
40   DIM R(25),Q(25,3),E(25,2),I(25,2),T(25,1),P(5,7)
50   DIM N(32,2),K(25,5),L(25,5),M(25,5),G(25,4)
60   DIM Y(30,30),V(31),U(31),C(30)
70   DIM A$[65],B$[9],D$[9],R$[25],C$[25],V$[25],I$[25],Q$[25],N$[5],H$[25]
80   R1=C1=V1=I1=Q1=M1=M2=I2=T9=N=T0=T2=T3=N(1,2)=M4=0
90   REDIM C(30)
100  B$="RVICQME*."
110  D$="AIDGOTPNS"
120  F=T4=1
130  DISP "INPUT CARD";
135 BEEP
140  INPUT A$
150  PRINT A$
160  FOR I=1 TO 9
170  IF A$[1;1]=B$[I;1] THEN 220
180  NEXT I
190  DISP "ILLEGAL CHARACTER: RE-";
200  GOTO 130
210  REM ...DETERMINE ELEMENT TYPE
220  ON I GOTO 240,350,560,670,790,850,1880,130,1290
230  REM ...RESISTORS
240  IF F=2 THEN 300
250  R1=T=R1+1
260  R$[R1]=A$[2;1]
270  GOSUB 2580
280  R(R1)=ABS(C(3))
290  GOTO 130
300  H$=R$
310  GOSUB 3030
320  R(T)=S
330  GOTO 130
340  REM ...VOLTAGE SOURCES
350  IF F=2 THEN 510
360  V1=T=V1+1
370  V$[V1]=A$[2;1]
380  GOSUB 2580
390  IF C(1)<>0 THEN 440
```

```
400  K(V1,I)=C(2)
410  L(V1,I)=0
420  E(V1,1)=-C(3)
430  GOTO 490
440  IF C(2)=0 THEN 480
450  PRINT "SOURCE UNGROUNDED: RE-";
460  V1=V1-1
470  GOTO 130
480  E(V1,1)=C(3)
490  E(V1,2)=C(4)
500  GOTO 130
510  H$=V$
520  GOSUB 3030
530  E(T,1)=S
540  GOTO 130
550  REM ...CURRENT SOURCES
560  IF F=2 THEN 630
570  I1=T=I1+1
580  I$[I1]=A$[2;1]
590  GOSUB 2580
600  I(I1,1)=C(3)
610  I(I1,2)=C(4)
620  GOTO 130
630  H$=I$
640  GOSUB 3030
650  I(T,1)=S1
660  GOTO 130
670  REM ...CAPACITORS
680  IF F=2 THEN 740
690  C1=T=C1+1
700  C$[C1]=A$[2;1]
710  GOSUB 2580
720  Q(T,1)=ABS(C(3))
730  GOTO 130
740  H$=C$
750  GOSUB 3030
760  Q(T,1)=S
770  GOTO 130
780  REM ...TRANSISTORS
790  IF F=2 THEN 190
800  Q1=T=Q1+1
810  Q$[Q1]=A$[2;1]
820  GOSUB 2580
830  T(Q1,1)=C(4)
840  GOTO 130
850  REM ...MODELS
860  IF F=2 THEN 1250
870  M1=T=M1+1
880  M$[T]=A$[2;1]
890  GOSUB 2580
900  FOR M=6 TO 9
```

```
910  IF A$[4;1]=D$[M;1] THEN 950
920  NEXT M
930  M1=M1-1
940  GOTO 190
950  FOR K=2 TO 7
960  P(T,K)=C(K-1)
970  NEXT K
980  ON M-5 GOTO 1080,990,1020,1040
990  IF A$[5;1]="U" THEN 1060
1000 P(T,1)=-1
1010 GOTO 1130
1020 P(T,1)=1
1030 GOTO 1130
1040 K=2
1050 GOTO 1100
1060 K=3
1070 GOTO 1100
1080 K=4
1090 T3=M1
1100 M2=M2+1
1110 P(T,1)=K
1120 GOTO 1210
1130 IF C(1)<>0 THEN 1150
1140 P(T,2)=100
1150 IF C(2)<>0 THEN 1170
1160 P(T,3)=1
1170 IF C(3)<>0 THEN 1190
1180 P(T,4)=1E-15
1190 IF C(4)<>0 THEN 1210
1200 P(T,5)=1E12
1210 S=VAL(M$[T;1])+10
1220 M(S,3)=T
1230 IF I=7 THEN 1880
1240 GOTO 130
1250 H$=M$
1260 GOSUB 3030
1270 GOTO 890
1280 REM ... CIRCUIT UPDATES
1290 IF F=1 THEN 190
1300 FOR J=1 TO 7
1310 IF A$[2;1]=D$[J;1] THEN 1340
1320 NEXT J
1330 GOTO 190
1340 F=J+1
1350 REDIM C(30)
1360 ON J GOTO 1380,1400,120,1510,1810,1410,1990
1370 REM ...ALTER
1380 GOTO 130
1390 REM ...INSERT
1400 GOTO 130
1410 IF A$[3;1]="R" THEN 1650
```

```
1420 REM ...TEMPERATURE
1430 IF T3=0 THEN 190
1440 PRINT "T(DEG C)";
1450 INPUT T1
1460 PRINT T1
1470 T1=T1+273
1480 T2=T1-300
1490 GOTO 130
1500 REM ...GAIN
1510 PRINT "INPUT";
1520 GOSUB 1540
1530 GOTO 1590
1540 PRINT " NODE";
1550 INPUT K
1560 PRINT K
1570 GOSUB 2970
1580 RETURN
1590 M=J
1600 PRINT "OUTPUT";
1610 GOSUB 1540
1620 PRINT "GAIN(V/V)=";Y(J,M)/Y(M,M)
1630 PRINT "INPUT IMPEDENCE=";Y(M,M)
1640 GOTO 130
1650 REM ...TRANSIENT
1660 F=8
1670 IF M4=0 THEN 1810
1680 PRINT "TIMESTEP=";"FINAL TIME=";
1690 INPUT D1,D9
1700 PRINT D1;D9
1710 PRINT "TIME";
1720 FOR I=1 TO M4
1730 L=M(I+9,4)
1731 X$="V"&VAL$(L)
1740 PRINT USING 1770;X$
1750 NEXT I
1760 PRINT USING 1770
1770 IMAGE   #,9X,4A,DDD
1775 PRINT
1780 T0=0
1790 GOTO 1950
1800 REM ...OUTPUT PRINT
1810 DISP "OUTPUT NODE";
1820 INPUT K
1830 PRINT "V";K
1840 M4=M4+1
1850 M(M4+9,4)=K
1860 IF F=8 THEN 1660
1870 GOTO 130
1880 IF Q1=0 THEN 1950
1890 IF M1-M2>0 THEN 1950
1900 M1=T=M1+1
```

```
1910 M$(M1)="0"
1920 MAT C=ZER
1930 M=8
1940 GOTO 950
1950 D=1E40
1960 N1=N-V1
1970 ON F GOTO 1980,4710,4160,130,7010,130,4710,4180
1980 REM ...PRINT INPUT DATA
1990 IF R1=0 THEN 2210
2000 PRINT
2010 PRINT "RESISTORS:"
2020 PRINT "NAME    NODES          VALUE"
2030 FOR I=1 TO R1
2040 PRINT "R";R$[I;1];TAB(6);K(I,1),L(I,1);R(I)
2050 NEXT I
2060 IF C1=0 THEN 2120
2070 PRINT "CAPACITORS:"
2080 PRINT "NAME    NODES          VALUE"
2090 FOR I=1 TO C1
2100 PRINT "C";C$[I;1];TAB(6);K(I,4);L(I,4);Q(I,1)
2110 NEXT I
2120 IF V1=0 THEN 2190
2130 PRINT
2140 PRINT "VOLTAGE SOURCES:"
2150 PRINT "NAME   +NODES-   VALUE  MODEL"
2160 FOR I=1 TO V1
2170 PRINT "V";V$[I;1];TAB(6);K(I,2);L(I,2);E(I,1);"M";E(I,2)
2180 NEXT I
2190 IF I1=0 THEN 2260
2200 PRINT
2210 PRINT "CURRENT SOURCES:"
2220 PRINT "NAME   +NODES-   VALUE MODEL"
2230 FOR I=1 TO I1
2240 PRINT "I";I$[I;1];TAB(6);K(I,3);L(I,3);I(I,1);"M";I(I,2)
2250 NEXT I
2260 IF Q1=0 THEN 2340
2270 PRINT
2280 PRINT "TRANSISTORS:"
2290 PRINT "NAME    C      B      E   MODEL"
2300 FOR I=1 TO Q1
2310 PRINT "Q";Q$[I;1];TAB(7);K(I,5);L(I,5);M(I,5);"M";T(I,1)
2320 NEXT I
2330 PRINT
2340 IF M1=0 THEN 2520
2350 PRINT "MODELS:"
2360 PRINT "NAME  TYPE"
2370 FOR I=1 TO M1
2380 J=ABS(P(I,1))
2390 ON J GOTO 2400,2440,2460,2480
2400 A$="NPN"
2410 IF P(I,1)=1 THEN 2490
```

```
2420 A$="PNP"
2430 GOTO 2490
2440 A$="SIN"
2450 GOTO 2490
2460 A$="PUL"
2470 GOTO 2490
2480 A$="TEM"
2490 PRINT USING 2500;M$[I;1],A$,P(I,2),P(I,3),P(I,4),P(I,5),P(I,6),P(I,7)
2500 IMAGE "M",1A,4X,3A,2(M5D.3D),M12D.3D,M12D.3D,M12D.3D,M12D.3D
2510 NEXT I
2520 PRINT
2530 PRINT "NODES:";N
2540 PRINT
2550 PRINT "****END OF INPUT DATA****"
2560 PRINT
2565 BEEP
2570 GOTO 3100
2580 REM ...SUB TO READ INPUT DATA
2590 J=8
2600 S=0
2610 IF I>5 THEN 2640
2620 S=POS(A$,"M")
2630 J=4
2640 K=0
2650 MAT C=ZER
2660 K=K+1
2670 L=POS(A$[J]," ")
2680 IF (J<S) OR (S=0) THEN 2710
2690 J=S+1
2700 GOTO 2750
2710 IF L=0 THEN 2750
2720 C(K)=VAL(A$[J,J+L-1])
2730 J=J+L
2740 GOTO 2660
2750 C(K)=VAL(A$[J])
2760 IF I=6 THEN 2950
2770 S=2
2780 IF I<>5 THEN 2800
2790 S=3
2800 FOR L=1 TO S
2810 IF C(L)=0 THEN 2880
2820 REM ...DET. UNIQUE NODE NUMBERS
2830 FOR M=1 TO N
2840 IF C(L)=N(M,2) THEN 2880
2850 NEXT M
2860 N=N+1
2870 N(N,2)=C(L)
2880 NEXT L
2890 K(T,I)=C(1)
2900 L(T,I)=C(2)
2910 IF I<>5 THEN 2930
```

```
2920 M(T,I)=C(3)
2930 IF F<>3 THEN 2950
2940 GOSUB 3590
2950 RETURN
2960 REM ...SUB TO DET. ELEMENT NODE
2970 FOR J=1 TO N
2980 IF N(J,2)=K THEN 3000
2990 NEXT J
3000 J=N(J,1)
3010 RETURN
3020 REM ...SUB TO FIND ALTER ELEMENT
3030 T=POS(H$,A$[2;1])
3040 IF T<>0 THEN 3070
3050 DISP "ELEMENT NOT FOUND;RE-";
3060 GOTO 130
3070 IF I=6 THEN 3090
3080 S=VAL(A$[4])
3090 RETURN
3100 REM ...PROCESS CIRCUIT DATA
3110 FOR I=1 TO N
3120 N(I,1)=I
3130 NEXT I
3140 REM ...REORDER NODE VECTOR
3150 FOR I=1 TO N
3160 FOR J=I+1 TO N
3170 IF N(I,2)<N(J,2) THEN 3210
3180 T=N(J,2)
3190 N(J,2)=N(I,2)
3200 N(I,2)=T
3210 NEXT J
3220 NEXT I
3230 REM ...MOVE SOURCE NODES TO END OF NODE VECTOR
3240 L=V1
3250 I=2
3260 GOSUB 3550
3270 M=N
3280 FOR I=1 TO V1
3290 K=K(I,2)
3300 IF K>N1 THEN 3400
3310 FOR L=1 TO V1
3320 IF L=I THEN 3360
3330 IF M<>K(L,2) THEN 3360
3340 M=M-1
3350 GOTO 3310
3360 NEXT L
3370 N(M,1)=K
3380 N(K,1)=M
3390 K(I,2)=M
3400 NEXT I
3410 REM ...RE-ORDER ELEMENT NODES
3420 L=R1
```

```
3430 I=1
3440 GOSUB 3550
3450 L=I1
3460 I=3
3470 GOSUB 3550
3480 L=C1
3490 I=4
3500 GOSUB 3550
3510 L=Q1
3520 I=5
3530 GOSUB 3550
3540 GOTO 3740
3550 FOR T=1 TO L
3560 GOSUB 3590
3570 NEXT T
3580 RETURN
3590 K=K(T,I)
3600 IF K=0 THEN 3630
3610 GOSUB 2970
3620 K(T,I)=J
3630 K=L(T,I)
3640 IF K=0 THEN 3670
3650 GOSUB 2970
3660 L(T,I)=J
3670 IF I<>5 THEN 3720
3680 K=M(T,I)
3690 IF K=0 THEN 3720
3700 GOSUB 2970
3710 M(T,I)=J
3720 RETURN
3730 REM ...REDUCE VOLTAGE SOURCES TO CURRENT EQUIVALENT FOR R AND C
3740 FOR I=1 TO V1
3750 J=K(I,2)
3760 REM ...RES.
3770 S=1
3780 S1=R1
3790 GOSUB 3900
3800 REM ...CURRENT SOURCE
3810 S=3
3820 S11=I1
3830 GOSUB 3900
3840 REM ..CAP.
3850 S=4
3860 S1=C1
3870 GOSUB 3900
3880 NEXT I
3890 GOTO 4110
3900 FOR M=1 TO S1
3910 K=K(M,S)
3920 L=L(M,S)
3930 IF J<>K THEN 3990
```

```
3940 IF S=4 THEN 3970
3950 K(M,S)=0
3960 IF S=3 THEN 4090
3970 T=L
3980 GOTO 4040
3990 IF J<>L THEN 4090
4000 IF S=4 THEN 4030
4010 L(M,S)=0
4020 IF S=3 THEN 4090
4030 T=K
4040 I2=I2+1
4050 M(I2,4)=S
4060 M(I2,3)=T
4070 M(I2,2)=I
4080 M(I2,1)=M
4090 NEXT M
4100 RETURN
4110 T1=300
4120 REM ..BEGIN ANALYSIS
4125 BEEP
4130 MAT V=ZER(N)
4140 MAT U=ZER(N)
4150 MAT G=ZER(Q1+1,4)
4160 IF F<>8 THEN 4710
4170 REM ...UPDATE TRANS. SOURCES
4180 IF T0<>D1 THEN 4210
4190 D=D1
4200 GOTO 4710
4210 FOR I=1 TO V1
4220 K=E(I,2)
4230 IF K=0 THEN 4260
4240 GOSUB 4340
4250 E(I,1)=V
4260 NEXT I
4270 FOR I=1 TO I1
4280 K=I(I,2)
4290 IF K=0 THEN 4320
4300 GOSUB 4340
4310 I(I,1)=V
4320 NEXT I
4330 GOTO 4710
4340 T=M(K+10,3)
4350 J=P(T,1)-1
4360 ON J GOTO 4370,4450
4370 REM ...SINE
4380 V=P(T,2)
4390 IF P(T,5)<>0 THEN 4410
4400 P(T,5)=D1
4410 IF T0<P(T,5) THEN 4430
4420 V=V+P(T,3)*SIN(2*PI*P(T,4)*(T0-P(T,5))+P(T,6)/57.296)
4430 RETURN
```

```
4440 REM ...PULSE
4450 Z=P(T,4)
4460 IF T0>Z THEN 4490
4470 V=P(T,2)
4480 RETURN
4490 Z=Z+P(T,5)
4500 IF T0>=Z THEN 4530
4510 V=P(T,3)-(P(T,3)-P(T,2))/P(T,5)*(Z-T0)
4520 RETURN
4530 Z=Z+P(T,6)
4540 IF T0>Z THEN 4570
4550 V=P(T,3)
4560 RETURN
4570 Z=Z+P(T,7)
4580 IF T0>=Z THEN 4610
4590 V=P(T,2)+(P(T,3)-P(T,2))/P(T,7)*(Z-T0)
4600 RETURN
4610 V=P(T,2)
4620 RETURN
4630 REM ...SUB TO DET. DELTA V
4640 V=0
4650 IF L=0 THEN 4670
4660 V=V(L)
4670 IF K=0 THEN 4690
4680 V=V-V(K)
4690 RETURN
4700 REM ...UPDATE CAPACITOR CURRENTS
4710 IF T3=0 THEN 4730
4720 T4=1+P(T3,4)*T2+P(T3,5)*T2^2
4730 FOR I=1 TO C1
4740 IF (F=8) AND (T0>0) THEN 4770
4750 Q(I,2)=Q(I,3)=0
4760 GOTO 4830
4770 K=K(I,4)
4780 L=L(I,4)
4790 GOSUB 4630
4800 T=Q(I,1)*T4/D*V
4810 Q(I,2)=-Q(I,3)-T
4820 Q(I,3)=T+Q(I,2)
4830 NEXT I
4840 REM ...ADD SUPPLIES TO V MATRIX
4850 FOR I=1 TO V1
4860 J=K(I,2)
4870 V(J)=E(I,1)
4880 NEXT I
4890 T9=0
4900 MAT Y=ZER(N1,N1)
4910 MAT C=ZER(N1)
4920 REM ...ADD RESISTORS
4930 IF T3=0 THEN 4950
4940 T4=1+P(T3,2)*T2+P(T3,3)*T2^2
```

```
4950 FOR I=1 TO R1
4960 K=K(I,1)
4970 L=L(I,1)
4980 R=1/R(I)/T4
4990 GOSUB 5020
5000 NEXT I
5010 GOTO 5100
5020 IF K=0 THEN 5070
5030 Y(K,K)=Y(K,K)+R
5040 IF L=0 THEN 5090
5050 Y(K,L)=Y(K,L)-R
5060 Y(L,K)=Y(L,K)-R
5070 IF L=0 THEN 5090
5080 Y(L,L)=Y(L,L)+R
5090 RETURN
5100 REM ...ADD CURRENT SOURCES
5110 FOR I=1 TO I1
5120 K=K(I,3)
5130 L=L(I,3)
5140 C=I(I,1)
5150 GOSUB 5180
5160 NEXT I
5170 GOTO 5230
5180 IF K=0 THEN 5200
5190 C(K)=C(K)+C
5200 IF L=0 THEN 5220
5210 C(L)=C(L)-C
5220 RETURN
5230 REM ...ADD CAPACITORS
5240 IF T3=0 THEN 5260
5250 T4=1+P(T3,4)*T2+P(T3,5)*T2^2
5260 FOR I=1 TO C1
5270 K=K(I,4)
5280 L=L(I,4)
5290 IF K<=N1 THEN 5310
5300 K=0
5310 IF L<=N1 THEN 5330
5320 L=0
5330 R=Q(I,1)*T4/D
5340 GOSUB 5020
5350 C=Q(I,2)
5360 GOSUB 5180
5370 NEXT I
5380 REM ...ADD GENERATED CURRENT SOURCES
5390 FOR I=1 TO I2
5400 J=M(I,1)
5410 K=M(I,2)
5420 L=M(I,3)
5430 IF M(I,4)=4 THEN 5460
5440 C(L)=C(L)+E(K,1)/R(J)
5450 GOTO 5470
```

```
5460 C(L)=C(L)+E(K,1)*Q(J,1)/D
5470 NEXT I
5480 REM ...ADD TRANSISTORS
5490 IF Q1=0 THEN 6450
5500 V6=8.6164E-5*T1
5510 C0=(T1/300)^3*EXP(-13920*(1/T1-1/300))
5520 IF T3=0 THEN 5540
5530 T4=1+T2*P(T3,6)+T2^2*P(T3,7)
5540 FOR I=1 TO Q1
5550 T=T(I,1)
5560 T=M(T+10,3)
5570 T7=P(T,1)
5580 REM ...INITIALIZE PARAMETERS FOR FIRST ITERATION
5590 IF T9<>0 THEN 5640
5600 S1=S2=0
5610 IF F<>0 THEN 5640
5620 G(I,1)=.5
5630 G(I,2)=0
5640 K=M(I,5)
5650 L=L(I,5)
5660 GOSUB 4630
5670 V4=V*T7
5680 M=K
5690 K=K(I,5)
5700 GOSUB 4630
5710 V3=V*T7
5720 Z=1
5730 IF V3>0 THEN 5750
5740 Z=Z-V3*T7/P(T,5)
5750 B=P(T,2)*T4*Z
5760 C2=C0*P(T,4)*(1+1/P(T,2))/(1+1/B)
5770 V=V4
5780 J=1
5790 GOSUB 5810
5800 GOTO 5960
5810 IF V<=G(I,J) THEN 5880
5820 C3=0
5830 IF G(I,J)<0 THEN 5850
5840 C3=C2*(EXP(G(I,J)/V6)-1)
5850 C6=C3-G(I,J+2)*G(I,J)
5860 A=(G(I,J+2)*V+C6)/C2+1
5870 V=V6*LOG(A)
5880 C3=-C2
5890 IF V<-2 THEN 5910
5900 C3=C2*EXP(V/V6)+C3
5910 G3=(C3+C2)/V6
5920 C6=T7*(C3-G3*V)
5930 G(I,J+2)=G3
5940 G(I,J)=V
5950 RETURN
5960 C4=C3
```

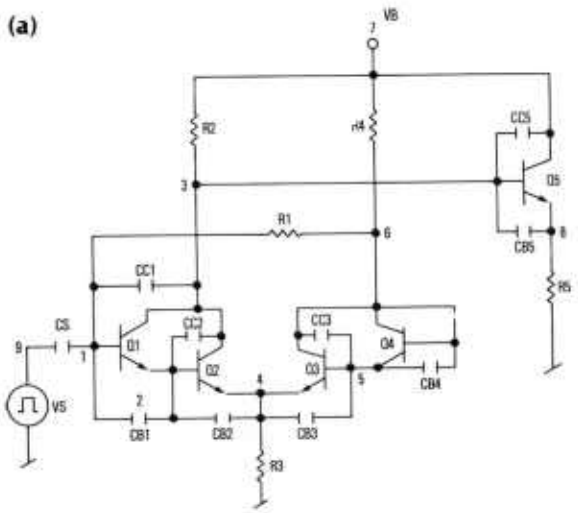
```
5970 C5=C6
5980 G1=G3
5990 B1=SQR(P(T,6)*C2)/P(T,2)
6000 C7=-B1
6010 IF V<-2 THEN 6030
6020 C7=B1*EXP(V/V6/2)+C7
6030 Z=(C7+B1)/V6/2
6040 C8=T7*(C7-Z*V)
6050 G2=G1/B+Z+C2
6060 B1=P(T,3)*T4
6070 C2=C0*P(T,4)*(1+1/P(T,3))/(1+1/B1)
6080 V=V3
6090 J=2
6100 GOSUB 5810
6110 G4=G3/B1+C2
6120 IF F<>5 THEN 6180
6130 C2=T7*(C4/B+C3/B1+C7)
6140 C4=T7*(C4-(1+1/B1)*C3)
6150 PRINT USING 6160;Q$[I;1],C2,C4,T7*V4,T7*V3,C4/C2,G1,1/G2
6160 IMAGE #,"Q",1A,2X,MD.DDDE,1X,MD.DDDE,2(3X,M2D.DDD),MDDD.DD,1X,MD.DDE,1X,MD.
DDE
6170 GOTO 6430
6180 REM ...GND. CONDUCTANCES AND V.D.C.D. CONNECTED TO SUPPLY
6190 IF K<=N1 THEN 6210
6195 C6=C6-G3*V(K)
6200 K=0
6210 IF L<=N1 THEN 6230
6215 C5=C5+G1*V(L)
6216 C6=C6+G3*V(L)
6220 L=0
6230 IF M<=N1 THEN 6250
6235 C5=C5-G1*V(M)
6240 M=0
6250 IF K=0 THEN 6340
6260 C(K)=C(K)+(1+1/B1)*C6-C5
6270 Y(K,K)=Y(K,K)+G3+G4
6280 IF L=0 THEN 6310
6290 Y(K,L)=Y(K,L)+G1-G3-G4
6300 Y(L,K)=Y(L,K)-G4
6310 IF M=0 THEN 6400
6320 Y(K,M)=Y(K,M)-G1
6330 Y(M,K)=Y(M,K)-G3
6340 IF M=0 THEN 6400
6350 C(M)=C(M)+(1+1/B)*C5-C6+C8
6360 Y(M,M)=Y(M,M)+G1+G2
6370 IF L=0 THEN 6430
6380 Y(M,L)=Y(M,L)-G1-G2+G3
6390 Y(L,M)=Y(L,M)-G2
6400 IF L=0 THEN 6430
6410 C(L)=C(L)-C5/B-C6/B1-C8
6420 Y(L,L)=Y(L,L)+G2+G4
```

```
6430 NEXT I
6440 IF F=5 THEN 130
6450 REDIM V(N1)
6460 MAT Y=INV(Y)
6470 MAT V=Y*C
6480 REDIM V(N)
6490 IF Q1=0 THEN 6700
6500 T9=T9+1
6510 REM ...CHECK FOR CONVERGENCE
6520 MAT U=V-U
6530 S=0
6540 FOR J=1 TO N1
6550 S=S+U(J)^2
6560 NEXT J
6570 IF F=8 THEN 6590
6580 PRINT S
6590 IF (S<N1^2*1E-10) AND (S1<N1^2*1E-10) AND (T9>3) THEN 6700
6600 IF (S>S1) AND (S1>S2) AND (T9>5) AND (S<.1) THEN 6640
6610 S2=S1
6620 S1=S
6630 GOTO 6680
6640 PRINT "MEAN ERROR(VOLTS):";SQR(S)
6650 PRINT
6660 GOTO 6700
6670 REM ...STORE LAST NODE VOLTAGES
6680 MAT U=V
6690 GOTO 4900
6700 IF F<>8 THEN 6810
6710 PRINT USING 6711;T0
6711 IMAGE  #,D.3DE
6720 FOR L=1 TO M4
6730 K=M(L+9,4)
6740 GOSUB 2970
6750 PRINT USING 6760;V(J)
6760 IMAGE #,  M4D.5D
6770 NEXT L
6780 PRINT USING 6781;T9
6781 IMAGE #,3D
6782 PRINT
6783 IF T0>D9 THEN 130
6790 T0=T0+D1
6791 Y9=Y9+T9
6800 GOTO 4180
6810 IF Q1=0 THEN 6840
6820 PRINT "ITERATIONS:";T9
6830 PRINT
6840 T=T1-273
6850 PRINT "T=";T,"DEG C"
6860 PRINT
6870 PRINT "NODE VOLTAGES:"
6880 FOR K=1 TO N
```

```
6890 I=N(K,1)
6900 J=N(K,2)
6910 PRINT USING 6920;J,V(I)
6920 IMAGE "V",M3D,M13D.4D
6930 NEXT K
6940 F=5
6950 IF Q1<>0 THEN 6970
6960 GOTO 130
6970 PRINT "TRANSISTOR OPERATING POINTS:"
6980 PRINT USING 6990
6990 IMAGE "NAME",5X,"IB",9X,"IC",9X,"VBE",7X,"VBC",6X,"BETA",6X,"GM",8X,"RPI"
7000 GOTO 5480
7010 STOP
7020 END
```

Four test circuits were used to compare the analytical speeds of BIAS-D modifications which are described in section 4 of the main body of the report. These test circuits are all modifications of the same test circuit used in section 3 to evaluate the BASIC version of BIAS-D. The basic circuit (CKT10) is shown in figure B-1(a). CKT10 is a nine-node, five-transistor integrated preamplifier circuit. Capacitors were added across the collector-base and base-emitter junctions of each transistor to represent the transistor junction capacitances. A BIAS-D input listing is given in figure B-1(b). CKT10 does not include any bulk resistor, but the other three circuits were obtained from CKT10 by successively adding resistors to the base (CKT11) (fig. B-2), collector (CKT12) (fig. B-3), and emitter (CKT13) (fig. B-4) of each transistor in this circuit.

(a)



(b)

```
* TEST CIRCUIT CKT10 (9 NODES)
**** INTEGRATED PREAMPLIFIER ****
* RESISTORS
R1 6 1 12K
R2 7 3 7.5K
R3 4 0 680
R4 7 6 9K
R5 8 0 5K
```

Figure B-1. Standard test circuit CKT10 (9 nodes): (a) diagram and (b) BIAS-D input listing.

```
* TRANSISTORS
Q1 3 1 2 M2
Q2 3 2 4 M2
Q3 6 5 4 M2
Q4 6 6 5 M2
Q5 7 3 8 M2
* VOLTAGE SOURCES
VS 9 0.1 M1
VB 7 0 6.1
* CAPACITORS
CS 9 1 1U
CB1 1 2 2P
CB2 2 4 2P
CB3 5 4 2P
CB5 3 8 2P
CC1 3 1 2P
CC2 3 2 2P
CC3 6 5 2P
CC5 7 3 2P
* MODELS
M1 PUL 0 -1 .5U .5U 5U .5U
M2 NPN 100 1 5E-15
END
* FOR BENCHMARK TIMES USE:
*       .TR
*       TR 0 10U .1U
*       V8 PRT
```

Figure B-1(b) (cont'd). Standard test circuit CKT10, BIAS-D input listing.

(a)

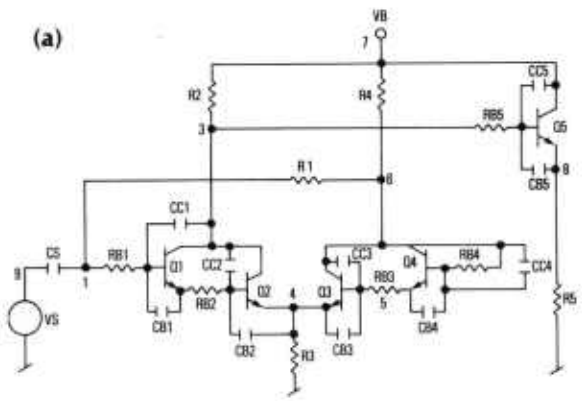


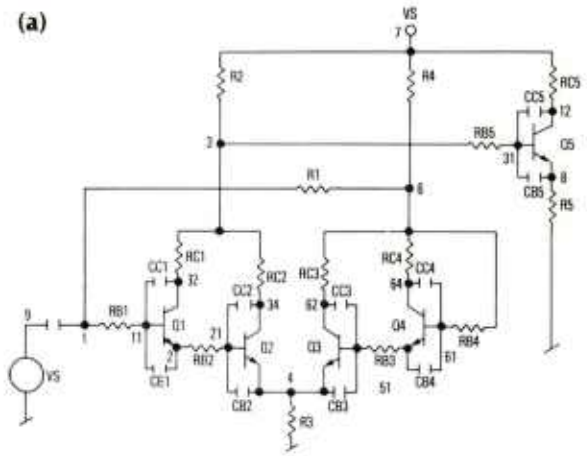
Figure B-2. Standard test circuit CKT11 (14 nodes): (a) diagram.

(b)

```
* TEST CIRCUIT CKT11 (14 NODES)
**** INTEGRATED PREAMPLIFIER ****
* RESISTORS
R1 6 1 12K
R2 7 3 7.5K
R3 4 0 680
R4 7 6 9K
R5 8 0 5K
* TRANSISTORS
Q1 3 11 2 M2
Q2 3 21 4 M2
Q3 6 51 4 M2
Q4 6 61 5 M2
Q5 7 31 8 M2
* VOLTAGE SOURCES
VS 9 0 1 M1
VB 7 0 6.1
*BASE RESISTORS
RB1 1 11 100
RB2 2 21 100
RB3 5 51 100
RB4 6 61 100
RB5 3 31 100
* CAPACITORS
CS 9 1 1U
CE1 11 2 2P
CC1 11 3 2P
CE2 21 3 2P
CC2 21 4 2P
CE3 51 6 2P
CC3 51 4 2P
CE4 61 5 2P
CC4 61 6 2P
CE5 31 7 2P
CC5 31 8 2P
* MODELS
M1 PUL 0 -1 .5U .5U 5U .5U
M2 NPN 100 1 5E-15
END
* FOR BENCHMARK TIMES USE:
*       .TR
*       TR 0 10U .1U
*       V8 PRT
```

Figure B-2. Standard test circuit CKT11 (14 nodes): (b) BIAS-D input listing.

(a)



(b)

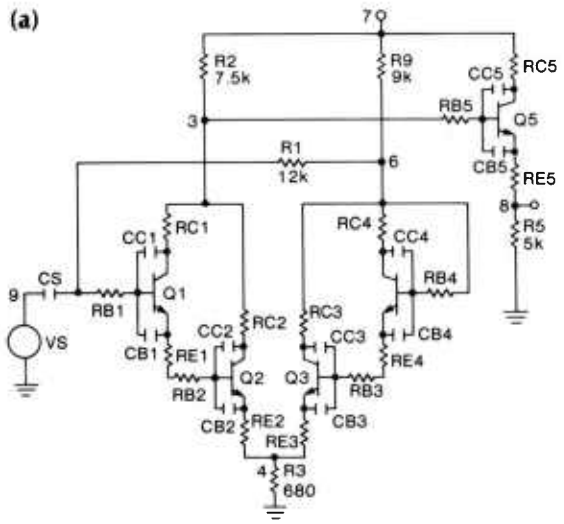
```
* TEST CIRCUIT CKT12 (19 NODES)
**** INTEGRATED PREAMPLIFIER ****
* RESISTORS
R1 6 1 12000
R2 7 3 7500
R3 4 0 680
R4 7 6 9000
R5 8 0 5000
* TRANSISTORS
Q1 32 11 2 M2
Q2 34 21 4 M2
Q3 62 51 4 M2
Q4 64 61 5 M2
Q5 72 31 8 M2
* VOLTAGE SOURCES
VB 7 0 6.1
VS 9 0 1 M1
CS 9 1 1U
*BASE RESISTORS
RB1 1 11 100
RB2 2 21 100
RB3 5 51 100
RB4 6 61 100
RB5 3 31 100
* COLLECTOR RESISTORS
RC1 3 32 100
RC2 3 34 100
RC3 6 62 100
RC4 6 64 100
RC5 7 72 100
```

Figure B-3. Standard test circuit CKT12 (19 nodes): (a) diagram and (b) BIAS-D input listing.

```
*JUNCTION CAPACITANCES
CE1 11 2 2P
CC1 11 3 2P
CE2 21 3 2P
CC2 21 4 2P
CE3 51 6 2P
CC3 51 4 2P
CE4 61 5 2P
CC4 61 6 2P
CE5 31 7 2P
CC5 31 8 2P
* MODELS
M1 PUL 0 -1 .5U .5U 5U .5U
M2 NPN 100 1 5E-15
END
* FOR BENCHMARK TIMES USE:
*       .TR
*       TR 0 10U .1U
*       U8 PRT
```

Figure B-3(b) (cont'd). Standard test circuit CKT12, BIAS-D input listing.

(a)



(b)

```
* TEST CIRCUIT CKT13 (24 NODES)
**** INTEGRATED PREAMPLIFIER ****
* RESISTORS
R1 6 1 12000
R2 7 3 7500
R3 4 0 680
R4 7 6 9000
R5 8 0 5000
* TRANSISTORS
Q1 32 11 23 M2
Q2 34 21 43 M2
Q3 62 51 44 M2
Q4 64 61 53 M2
Q5 72 31 83 M2
* VOLTAGE SOURCES
VB 7 0 6.1
VS 9 0 1 M1
CS 9 1 1U
*BASE RESISTORS
RB1 1 11 100
RB2 2 21 100
RB3 5 51 100
RB4 6 61 100
RB5 3 31 100
* COLLECTOR RESISTORS
RC1 3 32 100
RC2 3 34 100
RC3 6 62 100
RC4 6 64 100
RC5 7 72 100
* EMITTER RESISTORS
RE5 83 8 10
RE4 53 5 10
RE3 44 4 10
RE2 43 4 10
RE1 23 2 10
*JUNCTION CAPACITANCES
CE1 11 2 2P
CC1 11 3 2P
CE2 21 3 2P
CC2 21 4 2P
CE3 51 6 2P
CC3 51 4 2P
CE4 61 5 2P
CC4 61 6 2P
CE5 31 7 2P
CC5 31 8 2P
M1 PUL 0 -1 .5U .5U 5U .5U
M2 NPN 100 1 5E-15
END
* FOR BENCHMARK TIMES USE:
*       .TR
*       TR 0 10U .1U
*       U8 PRT
```

Figure B-4. Standard test circuit CKT13 (24 nodes): (a) diagram and (b) BIAS-D input listing.

Figure B-4(b) (cont'd). Standard test circuit CKT13, BIAS-D input listing.

APPENDIX C. BIAS-D LINKED-LIST STORAGE STRUCTURE

A linked-list storage structure is an efficient method of element storage in a circuit simulator in which a wide variety of circuits are to be analyzed. The linked element storage array used in BIAS-D (FORTRAN) resembles that used by Mini-MSINC.* Figure C-1 gives the BIAS-D configuration for each element list.

Four different list structures are shown here. Passive two-terminal elements (resistors, capacitors, or inductors) use the same list structure. Capacitors and inductors require two additional double-precision words for storage of temporary variables. Transistors use a similar configuration but reserve storage space for four single-precision temporary variables. Models use a different configuration. Here eight single-precision model parameters are stored with a pointer to a second list if necessary. The last list in this figure is that for the generated current sources. These sources are added during the setup procedure and are generated from the elements connected to voltage sources. All elements in this list point either to an element type or element value. They are stored sequentially and, therefore, do not need a pointer address. It is imperative that the list length for each element be divisible by two. This restriction enables simple addressing of integer and single-precision variables. This addressing is accomplished in BIAS-D as follows.

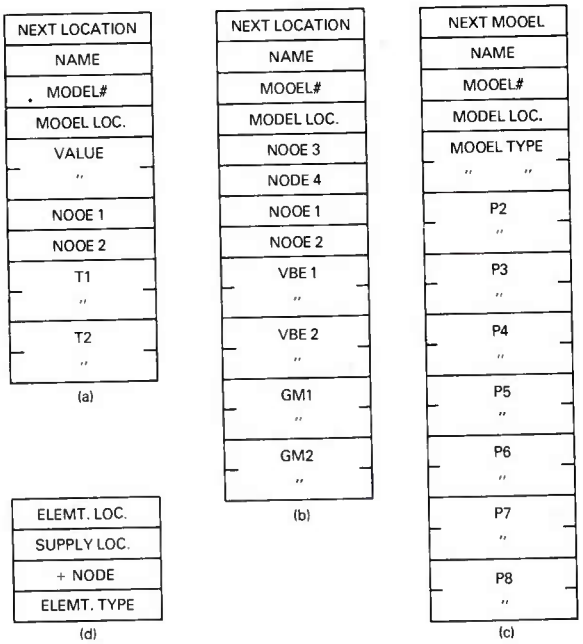


Figure C-1. Linked-list element storage array configuration in BIAS-D: (a) R, L, C, V, I elements, (b) transistors, (c) models, and (d) generated current sources.

integer location address = $K_{LOC} + L_{POS}$

single-precision location address = $K_{LOC}/2 + L_{POS}$,

where $K_{LOC} + 1$ is the integer location of the first variable in the particular element list, and L_{POS} is the displacement (in words) within this list. K_{LOC} is determined either from the IFRST array which gives the first location of each element type, or from the first location in each element list, which gives the address of the first location of the next element of that type. An extension of the previous restriction on the list configuration is that, within each element list, each single-precision variable must be on two word boundaries. This is for the same reason as given earlier. If the length of any element list is to be extended it can be easily done, in two-word increments, by changing the data statement in the MAIN subroutine containing the LEN variable. This variable defines the length in words of each element list.

If the particular computer system uses a two- or four-word double-precision data word format, double-precision data can also be stored in this list (not possible on the HP2100). In this case, each double-precision variable must be on boundaries equal to the data word length. The location of this varaible is found using the following address (for four-word double-precision data).

double-precision location address = $K_{LOC}/4 + L_{POS}$

T. K. Young and R. W. Dutton, Mini-MSINC—A Minicomputer Simulator for MOS Circuit with Modular Built-in Models, IEEE J. Solid-State Circuits, SC-11, No. 5, 730-732, October 1976.

There are no "PUSH" or "POP" routines in BIAS-D for loading or unloading this element list. Once the list is formed it is not changed, except when elements are inserted in the circuit. These are added at the end of the original element list, at starting location MXLOC. The generated current source pointers must then be regenerated and loaded at the new end of this list.

APPENDIX D. BIAS-D SUBROUTINE ORGANIZATION

The organization of the subroutines and functions contained in BIAS-D are described in this appendix. The BIAS-T9 version of BIAS-D is described. This version contains sparse matrix inversion and storage, linked-list element storage, and ac analysis using the traditional complex arithmetic method. A source listing of BIAS-T9 is given in appendix E.

The description of the subroutines is divided into four groups related by their function in BIAS-D. These groups are input/output, setup, analysis, and general functions. The relationships between these groups are shown in figure D-1. This shows the MAIN routine as controlling the entire input/output, setup, and analysis procedures with the general functions linked to all groups.

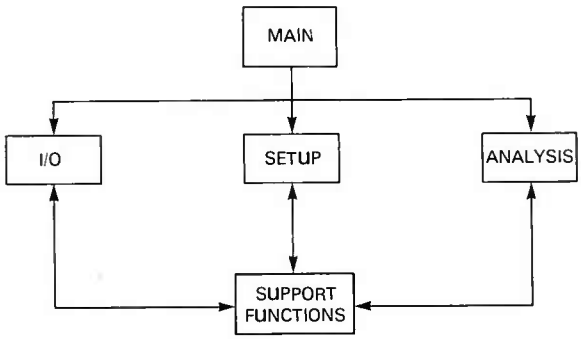


Figure D-1. Main subroutine groups in BIAS-D.

A more detailed flow diagram of the organization of the input/output group is given in figure D-2. A brief description of each of these routines is given as follows.

MCHEK checks for undefined element models. It stores the starting location of defined models, in the IELM array, with the appropriate element. M0 is the null model and is assigned to all elements with no user-defined model.

POUT sets up print or plot output formats. For print outputs, headings are printed for transient analyses, swept alter analysis, and ac analyses. The output device for printing is specified by the user.

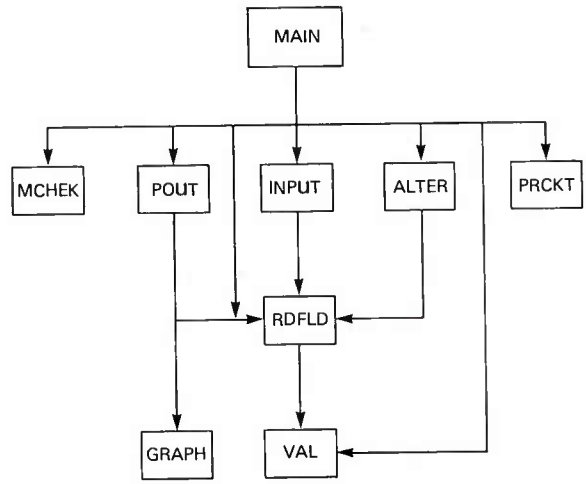


Figure D-2. Organization of input/output subroutines in BIAS-D.

This device can be the user terminal (TTY), disc, magnetic tape, or paper tape. For plot outputs, the graph axes are scaled and labeled for the appropriate analysis.

INPUT controls the initial reading of all element data from either a user terminal (TTY) or disc file. A limited amount of input processing is done in this routine. Unique node numbers are determined, and node numbers and element values are stored in the IELM array.

RDFLD reads a single floating-point and/or integer data field contained in the IAQ array. This field can contain up to eight floating-point or integer numbers separated by a comma or up to seven blanks. A pointer, LL, determines the starting location in the IAQ array of the decoding operation. A second pointer, KK, determines which number field within IAQ is being processed. The actual decoding of these numbers is done in function VAL.

VAL(LL) does the actual decoding of each number in the IAQ array. LL denotes the starting location of the number within the IAQ array. Any number may be preceded by as many as seven blanks and may be in one of several forms. For

example, the number one thousand may be represented as 1000, 1000.0, 1K, 1E3, 1E+3, or 1E 3.

ALTER is used to locate an element to be altered or an input source for a transient or ac analysis. It determines whether the element name being interrogated has been previously defined; if so, ALTER determines its beginning address in the IELM array so determined.

PRCKT writes the present circuit configuration in an ordered format to one of two output devices. During the initial dc analysis it is written to the user terminal. If called by the .SAVE command the same output configuration is written to a disc file. Then, this file can later be used as an input file. For this reason it is necessary that the format of the PRCKT output be readable by the INPUT routine.

Figure D-3 is an organizational diagram of the setup group of routines. In this group, program MAIN calls subroutine SETUP which in turn controls the setup procedure. A brief description of these routines follows.

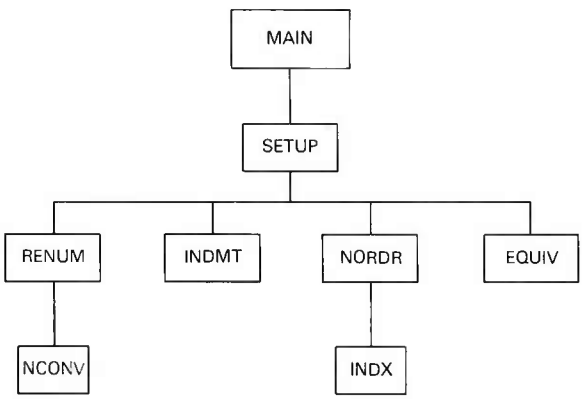


Figure D-3. Organization of setup subroutines in BIAS-D.

SETUP controls the entire setup procedure. It also renumbers the circuit nodes into a compact node set, and reorders these nodes such that the voltage source nodes are at the upper end of the node vector NI(i,2). NODE represents the total circuit nodes whereas NNODE gives the number of circuit nodes which are not connected to a voltage source.

RENUM(M) controls renumbering of the element node connections from the original node order to a compact node order determined in SETUP. The actual conversion is done in function NCONV. RENUM is called during the initial setup procedure and also if elements are to be inserted into the circuit using a .INSERT command (see app E).

NCONV(K,M,NI,NODE) returns a new node number, given a node number K. This is done by comparing input node K with the nodes in table NI(.,1) or NI(.,2) which are NODE nodes in length. If M = 0, K is converted from the original node number to a compact node number. If M = 1, K is converted from the compact node number to the original node number.

INDMT sets up an incidence matrix IY(i,j) for each new circuit. This integer matrix is then used to determine the optimum circuit node ordering.

NORDR determines the optimum node order for each circuit, and sets up the sparse matrix decomposition and storage pointers. The optimum order is obtained using the number of off-diagonal nonzero elements in the incidence matrix IY(i,j). This new node order is stored in vector IORDR. The row and column table locations of each nonzero matrix term used during the decomposition process is stored in arrays IUR and IUC. The location of each matrix entry used during an operation is stored in the IPOS array. The actual two-dimensional address generated during the decomposition procedure is converted into a linear address in function INDX(NR,NC).

INDX(NR,NC) converts a two-dimensional matrix address NR,NC into a location in the linear Y array. This is done by comparing row location NR and column location NC with the permitted table locations determined by the IUR and ILC pointer arrays.

EQUIV converts circuit voltage sources into Norton equivalent current sources. The number of these current sources generated depends on how many elements (and what type) are connected to

the voltage sources. The element type (resistor, capacitor, etc), the element location, the voltage source location, and the node into which the equivalent current source enters is stored in the IELM array at the end of the element linked list. This storage begins at location MXPOS.

A diagram of the organization of the analysis group of BIAS-D subroutines is given in figure D-4. Two main subroutine groups are controlled from subroutine ANALY and subroutine ACSOL. A brief description of those subroutines controlled from the ANALY group is given as follows.

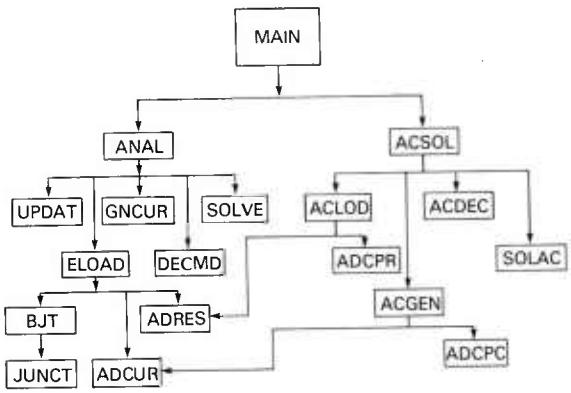


Figure D-4. Organization of analysis subroutines in BIAS-D.

ANALY controls the dc and transient analyses in BIAS-D. The capacitor and inductor currents are updated during a transient analysis, the current vector and admittance matrix are zeroed after each time-step, and convergence is determined for both dc and transient analyses.

UPDAT changes time-dependent voltage or current sources during a transient analysis. This new source value is stored in the IELM array to be later added to the voltage vector, V. The value of the time-dependent source is determined from the source model parameters also stored in the IELM array. The location in this array is stored with the particular source parameters and is determined during setup in the MCHEK routine.

ELOAD controls loading of equivalent real or imaginary element conductance values into the admittance matrix array, Y, and the equivalent currents into the current vector, C. For resistors, capacitors, and inductors, the actual loading of these conductance and current values is done in the ADRES and ADCUR routines for dc and transient analysis and in ADCPR and ADCPC for ac analysis. Bipolar transistors are loaded from the BJT subroutine.

ADRES adds a double-precision conductance value, DS, into the Y array at locations determined by element node numbers KK and LL as follows.

$$Y(KK,KK) = Y(KK) = Y(KK) + DS$$
$$Y(LL,LL) = Y(LL) = Y(LL) + DS$$
$$Y(KK,LL) = Y(JJ) = Y(JJ) - DS$$
$$Y(LL,KK) = Y(NN) = Y(NN) - DS$$

where JJ and NN are translated storage location values determined from function INDX(KK,LL).

ADCUR adds a double-precision current, DS, into the current vector, C, at locations determined from node values KK and LL as follows.

$$C(KK) = C(KK) - DS$$

$$C(LL) = C(LL) + DS$$

BJT determines the linearized Ebers-Moll equivalent conductance and current source values for bipolar transistors. The equivalent values for each junction, the collector-base, and the base-emitter are computed separately in subroutine JUNCT. These values are then added to the Y and C arrays.

JUNCT determines the Ebers-Moll junction equivalent conductance and current source values for a given junction voltage. Current or voltage update* is used depending on whether the new junction voltage is greater or less than the last junction voltage, respectively. The last junction voltage and transconductance for each transistor junction is stored in the IELM array.

**R. Barham, E. Cheung, and E. Cohen, BIAS-M, An Experimental Circuit Simulator for the IBM 1800, Integrated Circuits Group, University of California, Berkeley, June 1973.*

GNCUR determines the double-precision real or imaginary dependent current source values to be added into the current vector, C. These currents depend on the value of the independent voltage source value and the element values connected to these sources. The locations of these sources and elements are previously determined during setup by the EQUIV subroutine and stored in the IELM array.

DECMP does an LU decomposition of the sparse admittance matrix using the pointer structure generated during setup in subroutine NORDR.

SOLVE solves for the circuit node voltages using forward and backward substitution into the LU matrix generated in DECMP. The resulting node voltages are stored in double-precision vector V.

The second group of subroutines in the analysis portion of BIAS-D is controlled from subroutine ACSOL and is called during an ac analysis. Since complex arithmetic cannot be used, the admittance matrix is arranged so that all real entries are entered into the Y array and the imaginary entries into the YI array. The same pointer structure that was generated in NORDR and used in the dc and transient analysis is used to load both the Y and YI arrays. Resistors are added in the same manner as in previous analyses, using the ADRES subroutine. The imaginary conductance values of the capacitors and inductors at frequency FREQ are loaded into YI using subroutine ADCPR. A brief description of these routines follows.

ACSOL controls the small-signal ac frequency response analysis in BIAS-D. The ac parameters are initialized, the complex current vector and admittance matrix are zeroed for each new frequency point, and the next frequency point is determined.

BJTAC loads the ac bipolar transistor conductance values into the complex admittance matrix. All small-signal ac transistor conductance values are loaded into real array Y, since capacitors are not included in the transistor model.

ADCPR loads an imaginary double-precision conductance value, DS, into the imaginary part of the admittance matrix. The locations are determined in the same manner as in ADRES.

ADCPC adds an imaginary current, DS, into the CI array at locations KK and LL in the same manner as subroutine ADCUR.

DECAC does an LU decomposition of the complex sparse admittance matrix using the same pointer structure as used in DECMP and generated during setup in the NORDR subroutine.

SOLAC solves for the complex circuit node voltages using forward and backward substitution of the complex admittance matrix generated in DECAC. The resulting real node voltages are located in the V array and the imaginary voltages in the VI array.

APPENDIX E. BIAS-D USER'S MANUAL (FORTRAN VERSION) AND LISTING

E-1. INTRODUCTION

BIAS-D is a computer-aided circuit-analysis program written in FORTRAN IV for minicomputers with a minimum of 32 kwords of internal memory. It can perform ac, dc, and transient analysis of a 30-node circuit that contains up to 150 elements—resistors, capacitors, inductors, voltage sources, current sources, and transistors. For transistor circuits, BIAS-D converges to a solution by linearizing the built-in Ebers-Moll transistor model about an operating point in much the same manner as done in larger circuit-analysis programs such as BIAS-3, SLIC, and SPICE.

Circuit data are typed into the keyboard in a semifree input format. Error messages are given for recoverable data errors enabling immediate corrections. Transistor parameters, temperature coefficients, and transient sources are entered by specifying one or more of five available model types.

BIAS-D executes in a semi-interactive mode in which elements or models are altered, temperature varied, and elements inserted between existing nonsource nodes. The program is structured so that the circuit size and element capacity can be easily modified in accordance with the available memory size. Execution time for a dc solution of a 10-node, 5-transistor circuit is approximately 0.6 s on a PRIME 400 minicomputer.

E-2. INPUT DATA

The input data are divided into two categories: circuit data and control statement data. The circuit element data (e.g., resistors, transistors, etc) are input by specifying the element symbol (R, Q, M, etc) followed by the required data for that element. The control statement data are characterized by a dot (.) followed by the desired operation (.TRAN, .ALTER, etc). Control statements do not affect the results of the analyses—they only enable the user to direct the analysis procedure.

E-2.1 Circuit Data

Certain general instructions must be followed to input circuit data.

a. Each circuit element must begin in column 1.
b. Spaces are used as delimiters between data fields.
c. Scientific notation may be used (i.e., 1000 = 1E3).
d. Decimal points are not required (i.e., 2 = 2.0).
e. The ground node must be node 0 (zero).
f. Compact node numbering is not required (i.e., node numbers may be skipped).
g. The maximum allowable node number is 99.
h. Element values are to be in basic units (i.e., ohms, farads, volts, amperes, hertz, seconds).
i. Abbreviated notation may be used as follows:

$P = 10^{-12}$	$K = 10^{3}$
$N = 10^{-9}$	$ME = 10^{6}$
$U = 10^{-6}$	$G = 10^{12}$
$M = 10^{-3}$	

(e.g., 10U = 1.0E–5)

E-2.1.1 Resistors, Capacitors, Inductors

General form:

```
RXX N1 N2 VALUE M#
CXX N1 N2 VALUE M#
```

where XX is any two-character name, N1 and N2 are the node numbers (order not important), and VALUE is the resistor, capacitor, or inductor value in ohms, farads, or henries. The letter M followed by an integer from 1 to 9 denotes the model name (see sect. E-2.1.4). VALUE cannot be zero.

E-2.1.2 Independent Sources—Voltage, Current

General form:

```
VXX N+ N- VALUE M#
IXX N+ N- VALUE M#
```

where XX is any two-character name, N+ and N– are the positive and negative source nodes, respectively, and VALUE is the source value in volts or amperes. The letter M followed by an integer from 1 to 9 denotes the model name (see sect. E-2.1.4). For voltage sources, either N+ or N– must be grounded (node 0). For example,

```
V+ 3 0 5 M1
```

and

```
V+ 0 3 -5 M1
```

are equivalent.

For current sources, current flows from the positive node through the source to the negative node. The letter M followed by the model name may be omitted. However, a default number of zero will be assigned.

E-2.1.3 Bipolar Transistors

General form:

```
QXX NC NB NE M#
```

where XX is any two-character name, and NC, NB, and NE are the collector, base, and emitter node numbers, respectively. The letter M followed by an integer from 1 to 9 denotes the model name (see sect. E-2.1.4). The letter M followed by the model name may be omitted. However, a default number of zero (0) is assigned.

E-2.1.4 Models

General form:

```
M# YYY F1 F2 F3 F4 F5 F6
```

where # is an integer from 1 to 5 corresponding to the model number designated on the source or element. YYY is a three-letter name designating one of five available model types as follows.

1. NPN npn transistor parameters
2. PNP pnp transistor parameters
3. PUL pulse source specifications
4. SIN sinusoidal source specifications
5. EXT external source model
6. TEM element temperature coefficients

F1, F2, . . . , F6 are the data fields for specifying the above model parameters. These fields are defined below.

1. NPN—transistor parameters

Field	Parameter	Default value
F1	Forward dc beta (B_F)	100
F2	Reverse dc beta (B_R)	1
F3	Saturation current (I_S)	1E–15
F4	Early voltage (V_A)	1E+12
F5	Recombination current parameter (collector current at which beta = $B_F/2$)	0
F6	Not used	—

2. PNP—transistor parameters (same as NPN)

3. PUL—pulse source specifications

Field	Parameter	Default value
F1	Initial source value at t = 0	0
F2	Pulsed value	0
F3	Pulse delay time	T_{step}
F4	Pulse rise time	0
F5	Pulse duration (width)	0
F6	Pulse fall time	0

4. SIN—sinusoidal source specification

Field	Parameter	Default value
F1	dc source value (offset)	0
F2	Source amplitude (0-P)	0
F3	Source frequency (Hz)	0
F4	Time delay	T_{step}
F5	Phase shift (deg)	0
F6	Not used	—

The value of the sinusoidal source is determined by the equation

$$F(t) = F1 + F2 \cdot \sin[2\pi F3(t - F4) + F5].$$

5. EXT—External source parameters are to be defined by the user in a subroutine.

6. TEM—element temperature coefficients

Field	Temperature coefficient	Default value
F1	Resistor (T_{C1})	0
F2	Resistor (T_{C2})	0
F3	Capacitor (T_{C1})	0
F4	Capacitor (T_{C2})	0
F5	Transistor beta (T_{C1})	0
F6	Transistor beta (T_{C2})	0

The element value at temperature T is determined by the equation

$$E(T) = E(T_o)[1 + (T - T_o)T_{C1} + (T - T_o)^2 T_{C2}],$$

where T_o = 300 K. T_{C1} and T_{C2} are the element's first- and second-order temperature coefficients, respectively. The dimensions of T_{C1} and T_{C2} are in decimal percentages per degree Celsius (a decimal percentage of 0.002 = 2000 ppm/C).

E-2.1.5 Comment Statement

General form:

* any comment

A comment may be inserted at any line in the input circuit by using an asterisk (*) in column 1 followed by any message up to 80 characters long.

E-2.1.6 END statement

END terminates the inputting of circuit data. If a default transistor model is used, it may be necessary to use END twice in succession.

E-2.2 Control Commands

After completion of each type of analysis, program control is returned to the operator. This is indicated by "INPUT DATA" appearing on the display. At this time it is possible to initiate a new analysis. This is done by using one of the control commands described in the following sections; all control commands are prefixed by a dot (.).

E-2.2.1 .AC

The .AC command initiates the small-signal frequency response. This analysis can be obtained as follows.

.AC
"VIN FSTRT FSTOP PTS/DEC TYPE"

(enter "V"—input node, starting frequency, final frequency, frequency points per decade, and type of output; may also be current input—IIN)

"VXX PRT/PLO XMIN XMAX VMIN VMAX"

(enter "V"—output node, PRT—print, or PLT—plot)

For Print, no other parameters are necessary, and both the magnitude gain (TYPE = 0) or decibel gain (TYPE = 1) and phase of node XX are printed. For Plot, X and Y scale parameters are necessary (defaults are used if none are given). The plot type is determined by the value of TYPE.

TYPE = 0—magnitude gain
1—decibel gain
2—phase

E-2.2.2 .ALTER

The .ALTER command enables element values, models, and model parameters to be altered. This is done as follows.

```
.ALTER
RXX  VALUE
VXX  VALUE
  .
  .
  .
END
```

where XX is a valid element name (i.e., has been previously defined) and VALUE is the new element value. One or more element values may be altered using a single .ALTER command. An END statement terminates the alter operation. Models and model parameters may be altered in the same manner as the elements. Model types may be changed by entering a different model designation (see sect. E-2.1.4). For example, a pulse source PUL can be changed to a sinusoidal source, SIN, etc. All model parameters must be entered or they will be set to their default values. Both models and elements can be altered at the same time.

An additional .ALTER command permits sweeping element values over a specified range of values. This can be done as follows.

```
.ALTER
VXX EI EF DEL
```

where EI is the initial element value, EF the final element value, and DEL the increment value (DEL can be negative). This must be the last statement in a .ALTER command. It is then necessary to define an output node, a PRT/PLT specification, and so on (see sect. E-2.2.1). At the end of this analysis the altered value is returned to its original value.

E-2.2.3 .END

The .END command permits entering a new circuit without terminating the program. At this time all previous circuit values, names, and nodes are erased from memory.

E-2.2.4 .INSERT

The .INSERT command permits elements, models, or additional nodes to be inserted into an existing circuit. Any element or model may be inserted with this command. The .INSERT command is used as follows.

```
.INSERT
RXX N1 N2 VALUE
QXX NC NB NE M#
M# YYY F1 F2 F3 F4 F5 F6
      .
      .
      .
END
```

The format for the elements and models is the same as described at the beginning of section E-2.

E-2.2.5 .LOAD

The .LOAD command permits loading a circuit directly from a disc file. This is done as follows.

```
.LOAD
"ENTER FILENAME"
(enter file name)
```

Several circuits may be merged or models entered by successively using the .LOAD command. This command is terminated by an END statement (either in a file or via keyboard). Note that when several circuits are merged, unique node numbering must be maintained.

E-2.2.6 .PRINT

The element names and values can be displayed at any time by using the .PRINT command. Note that the node numbers displayed are a correct set of node numbers but are not necessarily the original set of numbers. If the original set of node numbers is necessary, the following sequence of commands can be used.

```
.INSERT
.PRINT
```

E-2.2.7 .SAVE

The .SAVE command is similar to the .LOAD command except that the circuit is written to a disc file. The contents of this file will be identical to that printed by a .PRINT command.

E-2.2.8 .TEMP

The analysis of the circuit at a temperature other than 27 C is obtained as follows.

```
.TEMP
"T(DEG C)"
(enter temperature)
```

This procedure is repeated for each new temperature. If a TEM model has not been defined, "TEMP. MODEL NOT SPECIFIED" will be displayed. This model can be inserted with the .INSERT command. Note that any subsequent analysis will be performed at the last temperature specified.

E-2.2.9. TRAN

A transient analysis can be obtained using the .TRAN command as follows.

```
.TRAN
"TR" T0 TF TSTEP
(enter "TR" to Tstep)
```

where T0 is the initial transient time, TF the final transient time, and TSTEP the output time increment. In order for the transient analysis to be meaningful, one or more source models (SIN, PUL, EXT) must have been specified. Voltage or current sources as well as models can be inserted once the initial circuit has been entered (see sect. E-2.2.4).

Note: Any control command (except .LOAD and .SAVE) will override a previously initiated control command. If a reply is expected, the command should be entered twice; the first time will cause an error message which can be ignored.

E-3. MISCELLANEOUS

Sometimes convergence to the desired accuracy is not attained. If this happens a "MEAN ERROR" message will appear. These results may or may not be correct. If, during a dc analysis, a more accurate solution is desired, the following procedure can be used.

```
.ALTER
END
```

This does not change the circuit but does allow at least four more iterations to occur.

In the general version of BIAS-D, several system-dependent subroutines have been commented out. These routines are OPNFL, CLSFL, GRAPH, IPACK, and SECND. Although BIAS-D will run without these routines, their implementation is desirable. A summary of these subroutine functions is as follows.

OPNFL CLSFL	Permits storage and retrieval of disc files.
GRAPH	Permits graphical output on any refresh graphics or storage tube graphic terminal.
IPACK	Permits use of two-character element names.
SECND	Gives elapsed execution times.

E-4. BIAS-D SOURCE LISTING (FORTRAN)

A listing of the FORTRAN version of BIAS-D is given here. This version of BIAS-D will run on a PDP 11/45, an HP2100 (HP21MX), a PRIME 400, and an IBM 370/168 with few source code changes. These changes are primarily concerned with individual computer system features such as timing, file management, etc.

APPENDIX E

```
C     ** CBIAS0 *************  BIAS-D  ************************
C     MINICOMPUTER AIDED ELECTRONIC CIRCUIT ANALYSIS PROGRAM
C         BIAST10 (*TEST10)     10-11-77
C          UPDATED 11-4-77,  10-3-78
C         DYNAMIC ELEMENT ALLOCATION  (LINKED-LIST)
C         DOUBLE PRECISION LU DECOMPOSITION
C         AC ANALYSIS- STANDARD METHOD USING COMPLEX MATRIX
C         SPARSE MATRIX INVERSION
C         ELEMENT MODELS
C         SPARSE STORAGE OF Y MATRIX
C
C      PROGRAM BIAS-D IS AVAILABLE AT NO CHARGE.  THE WORD ORIENTED
C      STRUCTURE OF BIAS-D PERMITS IT TO BE RUN ON ANY COMPUTER
C      SYSTEM SUPPORTING ANSII STANDARD FORTRAN IV WITH CAPABILITY OF
C        REAL/INTEGER WORDSIZE RATIO OF TWO.
C      INQUIRES SHOULD BE SENT TO THE AUTHOR:
C
C                        BRIAN L. BIEHL
C                      HARRY DIAMOND LABS
C                      2800 POWDER MILL RD.
C                      ADELPHI, MD. 20783
C                       (202) 394-3192
C
C     *******************************************************
C
      INTEGER VI,QI
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IBQ(12),IMQ(6),IDQ(8),IDATE(3),LEN(9)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(6),V1),(IELN(7),M1),(IELN(4),Q1)
      DATA IBQ/1HR,1HC,1HL,1HQ,1HI,1HV,1HM,1HE,1H*,1H.,1H+,1H /
      DATA IDQ/1HA,1HI,1HP,1HT,1HE,1HS,1HG,1HL/
      DATA IMQ/1HN,1HT,1HS,1HE,1HP,1HU/
      DATA LEN/8,12,12,16,8,8,20,0,0/
C
C          IFLG VALUES                                  IEL VALUES
C 1- INITIAL DC ANALYSIS 6- SWEPT ALTER          1-RESISTOR      5-CURRENT SOURCE
C 2- ALTER               7- SAVE CIRCUIT         2-CAPACITOR     6-VOLTAGE SOURCE
C 3- INSERT              8- SMALL SIGNAL GAIN    3-INDUCTOR      7-MODEL
C 4- PRINT CIRCUIT       9- AC ANALYSIS          4-BJT
C 5- TRANSIENT ANALYSIS 10- TEMPERATURE ANALYSIS
C
 100   CALL INITL
       CALL CLOCK(ETIM,IDATE)
       WRITE(IW,101) IDATE
 101   FORMAT(1X,10(1H*),21H  BIAS-D (11-04-77)  ,10(1H*),
      8 6HDATE: ,A2,1H/,A2,2H/7,A1/20X,5(1H-),8H TEST10 ,5(1H-)//)
 140   WRITE(IW,141)
 141   FORMAT(11H INPUT DATA)
       ITOTL=0
       NUNIT=IR
       CALL CLOCK(ETIM,IDATE)
       CALL SECND(TIM1)
 150   READ(NUNIT,151) IAQ
 151   FORMAT(80A1)
       LL=1
        IF(IAQ(1).EQ.IBQ(12)) LL=2
       DO 180 IEL=1,12
       IF(IAQ(LL).EQ.IBQ(IEL))GO TO 230
 180   CONTINUE
 200   WRITE(IW,201)
 201   FORMAT(23H ILLEGAL CHARACTER: RE-)
       GO TO 140
 220   IFLG=4
       GO TO 140
C .... DETERMINE ELEMENT TYPE
```

```
230   IF(IEL.LT.8)GO TO 260
      I=IEL-7
      GO TO(1840,150,1290,150,150),I
260   IF(IFLG.EQ.2)GO TO 1380
      ITT=LPOS+1
      CALL INPUT
      ILAST(IEL)=LPOS
      IF(LPOS.LT.MXLST) GO TO 280
270   WRITE(IW,271)
271   FORMAT(23H ELEMENT ARRAY OVERFLOW )
      GO TO 220
280   IF(IEL.EQ.7) GO TO 890
      LPOS=LPOS+LEN(IEL)
C .... REPROCESS UNGROUNDED OR NEGATIVE VOLTAGE SOURCES
300   IF(IEL.NE.6)GO TO 150
      IF(A(1).NE.0.)GO TO 450
      IELM(ITT+6)=A(2)
      IELM(ITT+7)=0
      ITT=ITT/2
      RELM(ITT+3)=-A(3)
      GO TO 150
450   IF(A(2).EQ.0.)GO TO 150
      WRITE(IW,461)
461   FORMAT( 23H SOURCE UNGROUNDED: RE-)
      V1=V1-I
      GO TO 140
C .... PROCESS MODELS
890   ITT=LPOS
      MNUM=VAL(LL+1)
      IELM(LPOS+3)=MNUM
C .... ENTRY POINT FOR ALTERD MODEL
C .... SKIP LEADING BLANKS
900   IF(IAQ(LL).NE.IBQ(12))GO TO 910
      LL=LL+I
      IF(LL.GT.12) GO TO 200
      GO TO 900
910   I=LL
      LL=LL+4
      CALL RDFLD
      IF(MM.LT.0) GO TO 940
C .... CHECK IF LEGAL MODEL TYPE
      DO 930 M=1,5
      IF(IAQ(I).EQ.IMQ(M))GO TO 960
930   CONTINUE
940   IF(IFLG.NE.2) M1=M1-I
      GO TO 200
960   MPOS=ITT/2
      DO 970 K=1,7
      KK=MPOS+K+3
      RELM(KK)=A(K)
970   CONTINUE
C .... DETERMINE MODEL TYPE
      I=1+I
      IF(M.LT.5)GO TO 1000
      M=6
      IF(IAQ(I).EQ.IMQ(6))M=5
      IF(IAQ(I).EQ.IMQ(1))M=-I
1000  RELM(MPOS+3)=M
      IF(M.EQ.-1)GO TO 1140
      IF(M.EQ.1) GO TO 1140
      IF(M.EQ.2)ITEMP=ITT/2
      GO TO 1240
C ....  BJT MODEL DEFAULT PARAMETERS
1140  IF(A(1).EQ.0.) RELM(MPOS+4)=100.
      IF(A(2).EQ.0.) RELM(MPOS+5)=1.0
      IF(A(3).EQ.0.) RELM(MPOS+6)=1.0E-15
      IF(A(4).EQ.0.) RELM(MPOS+7)=1.0E12
1240  IF(IFLG.EQ.2) GO TO 150
      LPOS=LPOS+LEN(7)
      GO TO 150
C
C   ****CIRCUIT UPDATES****
1290  IF(IAQ(LL+I).NE.IDQ(8)) GO TO 1300
C
C   ***.LOAD
      CALL OPNFL(IDISC,IW,IR)
      NUNIT=IDISC
      GO TO 150
1300  IF(IFLG.EQ.1)GO TO 200
```

APPENDIX E

```
C .... DETERMINE UPDATE TYPE
      DO 1320 J=1,7
      IF(IAQ(LL+1).EQ.IDQ(J))GO TO 1340
1320  CONTINUE
      GO TO 200
1340  IFLG=J+1
      GO TO (1360,1400,1460,1410,100,1470,1500),J
C
C  ***.AC
1360  IF(IAQ(LL+2).NE.IBQ(2))GO TO 140
      IFLG=9
      WRITE(IW,1361)
1361  FORMAT(24H VIN FSTRT FSTOP PTS/DEC )
      READ(IR,151) IAQ
      IEL=6
      IF(IAQ(LL).EQ.IBQ(5)) IEL=5
      CALL ALTER
      IF(IERR.EQ.0) GO TO 220
C .... INPT CONTAINS STARTING LOCATION OF SOURCE VALUE IN IELN()
      INPT=ITT/2
      DO 1370 M=1,4
      TM(M)=A(M)
1370  CONTINUE
      GO TO 1665
C
C   ***.ALTER
1380   CALL ALTER
       IF(IERR.EQ.0)GO TO 140
       IF(IEL.EQ.7) GO TO 900
       MPOS=ITT/2
       TM(5)=RELM(MPOS+3)
       RELM(MPOS+3)=A(1)
       IF(IEL.EQ.4)GO TO 200
       IF(A(3).EQ.0.)GO TO 150
C .... PROCESS SWEPTED ALTER
       DO 1390 M=1,3
       TM(M)=A(M)
1390   CONTINUE
       TM(4)=MPOS
       LTYPE=IEL
       IFLG=6
       GO TO 1665
C
C   ***.INSERT
1400   IF(JFLG.EQ.1) GO TO 140
       CALL RENUM(I)
       IELN(9)=0
       GO TO 140
C
C  ***.TEMPERATURE
1410   IF(IAQ(LL+2).EQ.IBQ(1))GO TO 1620
       IF(ITEMP.NE.0)GO TO 1440
       WRITE(IW,1421)
1421   FORMAT(31H TEMP. MODEL NOT SPECIFIED**RE-)
       GO TO 140
1440   WRITE(IW,1441)
1441   FORMAT( 9H T(DEG C))
       READ(IR,151) IAQ
       IFLG=10
       TEMP=VAL(I)+273.
       DTEMP=TEMP-300.0
       GO TO 140
C
C ***.PRINT CKT
1460   CALL PRCKT
       GO TO 140
C
C ***. SAVE CKT ON DISC FILE
1470   CALL OPNFL(IDISC,IW,IR)
       IUNIT=IDISC
       CALL PRCKT
       CALL CLSFL(IDISC)
       IUNIT=IW
       GO TO 220
C
C   ***.GAIN
1500   GO TO 220
C .... NOT IMPLEMENTED IN THIS VERSION
```

```
C
C   ***.TRANSIENT
1620  WRITE(IW,1631)
1631  FORMAT(20H "TR" T0 TSTOP TSTEP)
      READ(IR,151)IAQ
      LL=4
      CALL RDFLD
      IF(IERR.EQ.-1)GO TO 220
      IF(A(3).EQ.0.)A(3)=(A(2)-A(1))/50.
      DELTA=A(3)
      TM(2)=A(2)
      T0=A(1)
1665  CALL POUT
      JOUT=NCONV(IOUT,0,NI,NODE)
      GO TO 4190
C
1840  IF(NUNIT.EQ.IDISC) CALL CLSFL(IDISC)
      IF(IFLG.EQ.2) GO TO 4190
      IF(IFLG.GE.5) GO TO 4190
      IF(IFLG.EQ.4) GO TO 5200
      CALL MCHEK
      IF(IERR.EQ.0)GO TO 220
1910  NNODE=NODE-VI
      IF(NNODE.LE.NDMAX)GO TO 1950
      WRITE(IW,1921)
1921  FORMAT(20H NODE LIMIT EXCEEDED)
      GO TO 220
C .... CHECK FOR UNCONNECTED NODES
1950  J=1
      DO 2480 I=I,NODE
      IF(NI(I,1).GT.0) GO TO 2480
      WRITE(IUNIT,1961) NI(I,2)
1961  FORMAT(28H ONLY ONE CONNECTION AT NODE,I2)
      J=0
2480  CONTINUE
      IF(J.EQ.0) GO TO 220
      CALL POUT
      WRITE(IUNIT,101)  IDATE
      CALL PRCKT
      WRITE(IW,2491) NODE
2491  FORMAT(7H NODES:,I4)
      WRITE(IUNIT,2511)
2511  FORMAT(//28H **** END OF INPUT DATA ****//)
C
      CALL SECND(TIM2)
      CALL SETUP
      IF(IERR.EQ.-2) GO TO 270
      TEMP=300.
      DO 4180 I=1,NNODE
      V(I)=0.D0
      U(I)=0.D0
4180  CONTINUE
4190  DELT=1.0D12
      CALL SECND(TIM3)
      IF(IFLG.NE.9) GO TO 4500
      CALL ACSOL
      GO TO 4600
4500  CALL ANALY
4600  CALL SECND(TIM4)
      CALL CLOCK(ETIM2,IDATE)
      IF(IPLT.EQ.0)GO TO 4700
C      CALL EXITGR
C      K=IWAITQ(2)
      IPLT=0
4700  TIM1=TIM2-TIM1
      TIM2=TIM3-TIM2
      TIM3=TIM4-TIM3
      ETIM=ETIM2-ETIM
      IF(JFLG.EQ.1) WRITE(IUNIT,4801)TIM1,TIM2
4801  FORMAT(//6X,18H READIN TIME(SEC):,F12.3/
     8 7X,17H SETUP TIME(SEC):,F12.3)
      WRITE(IUNIT,4901) TIM3,ETIM
4901  FORMAT(4X,20H ANALYSIS TIME(SEC):,F12.3//
     8 25H TOTAL ELAPSED TIME(SEC):,F12.3/)
      IF(JFLG.NE.1)WRITE(IUNIT,5001)ITOTL
5001  FORMAT(18H TOTAL ITERATIONS=,I10)
      T0=0.
      JFLG=IFLG
      IF(IUNIT.EQ.IDISC) CALL CLSFL(IDISC)
```

```
      IUNIT=IW
      GO TO 220
5200  STOP
      END
      FUNCTION INDX(NR,NC)
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(30),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1))
C
C .... DETERMINE LINEAR Y ADDRESS LOCATION FROM Y(I,J)
C        Y(.,.) ADDRESS
C       MODIFIED FROM SINC-S 6-6-77
      IF(NR.EQ.NC) GO TO 190
      IS=NCON(NR)
      JS=NCON(NC)
      IF(JS.GT.IS) GO TO 130
C .... LOWER TRIANGLE
      N=ILC(JS)
      NE=ILC(JS+1)
115   IF(N.GT.NE) GO TO 185
      IF(NR.EQ.ILR(N)) GO TO 125
      N=N+1
      GO TO 115
125   INDX=N+NODE
      NN=N+MLOC
      RETURN
C .... UPPER TRIANGLE
130   N=IUR(IS)
      NE=IUR(IS+1)
135   IF(N.GT.NE) GO TO 185
      IF(NC.EQ.IUC(N)) GO TO 145
      N=N+1
      GO TO 135
145   INDX=N+MLOC
      NN=N+NODE
185   RETURN
C .... DIAGONAL LOCATION
190   INDX=NR
      NN=NC
      RETURN
      END
      SUBROUTINE INITL
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(30),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
C .... INITIALIZE READ/WRITE UNITS
      IW=1
      IR=1
      IDISC=5
      IUNIT=IW
      LPOS=0
      MXLST=1000
```

```
      NDMAX=30
C .... INITIALIZE ELEMENT COUNTERS
      DO 110 K=1,9
      IELN(K)=0
110   CONTINUE
      DO 140 K=1,MXLST
      IELM(K)=0
140   CONTINUE
      NODE=0
      ITEMP=0
      DTEMP=0.
      ITER=0
      IPLT=0
      NI(1,2)=0
      IFLG=1
      JFLG=1
      T0=0.
      RETURN
      END
      SUBROUTINE MCHEK
C .... CHECK FOR UNDEFINED MODELS AND STORE
C        LOCATION OF MODEL WITH ELEMENT
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(7),M1)
C
      IERR=1
      DO 300 IEL=1,6
      K1=IELN(IEL)
      IF(K1.EQ.0) GO TO 300
      KPOS=IFRST(IEL)
      DO 200 J=1,K1
      M=IELM(KPOS+3)
      IF(M.EQ.0) GO TO 190
      MPOS=IFRST(7)
      DO 50 K=1,M1
      N=IELM(MPOS+3)
      IF(N.NE.M) GO TO 40
      IELM(KPOS+4)=MPOS/2
      GO TO 190
40    MPOS=IELM(MPOS+1)
50    CONTINUE
95    WRITE(IW,91)M
91    FORMAT(9H MODEL: M,I1,12H NOT DEFINED)
      IERR=0
190   KPOS=IELM(KPOS+1)
200   CONTINUE
300   CONTINUE
      RETURN
      END
      SUBROUTINE INPUT
C .... CONTROLS READING OF INPUT DATA
      INTEGER*2 IELM,LHALF
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1)),(L,LHALF)
```

APPENDIX E

```
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      K1=IELN(IEL)+1
      IF(K1.GT.1) GO TO 2400
      IFRST(IEL)=LPOS
      GO TO 2500
2400  KPOS=ILAST(IEL)
      IELM(KPOS+1)=LPOS
2500  L=IPACK(IAQ,LL+1)
      IELM(LPOS+2)=LHALF
      IF(IEL.EQ.7) GO TO 3400
C .... READ INPUT DATA
      LL=LL+3
2600  CALL RDFLD
      IF(MM.GE.0) GO TO 2700
      K1=K1-1
      GO TO 3400
2700  IS=2
      IF(IEL.NE.4) GO TO 2800
      IS=3
2800  DO 3100 L=1,IS
      II=A(L)
      IF(II.EQ.0)GO TO 3100
C .... DETERMINE UNIQUE NODE NUMBERS
      DO 2900 M=1,NODE
      IF(II.EQ.NI(M,2))GO TO 3000
2900  CONTINUE
      NODE=NODE+1
      NI(NODE,2)=II
      GO TO 3100
3000  NI(M,1)=1
3100  CONTINUE
      IELM(LPOS+7)=A(1)
      IELM(LPOS+8)=A(2)
      IF(IEL.NE.4)GO TO 3200
      IELM(LPOS+5)=A(3)
      GO TO 3350
3200  IF(A(3).GT.0.)GO TO 3300
      IF(IEL.LT.5)A(3)=-A(3)
3300  MPOS=LPOS/2
      RELM(MPOS+3)=A(3)
3350  IELM(LPOS+3)=A(4)
3400  IELN(IEL)=K1
      RETURN
      END
      SUBROUTINE RDFLD
C .... READ DATA FIELD
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
C .... KK IS FIELD POINTER
C      LL IS COLUMN POINTER
      DO 1000 KK=1,8
      A(KK)=0.
1000  CONTINUE
      DO 1100 KK=1,8
      A(KK)=VAL(LL)
      IF(MM.LE.1)GO TO 1200
      LL=LL+1
1100  CONTINUE
1200  RETURN
      END
      SUBROUTINE PRCKT
C .... PRINT INPUT DATA
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
```

```
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION ITYPE(14),NAME(10)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      DATA ITYPE/2H P,2HNP,0,0,2H N,2HPN,2H T,2HEM,2H S,2HIN,2H E,
     I 2HXT,2H P,2HUL/
      DATA NAME/1HR,1HC,1HL,1HQ,1HI,1HV,1HM,1H ,1H+,1H-/
C
      DO 300 I=1,7
      K1=IELN(I)
      IF(K1.EQ.0)GO TO 300
      KPOS=IFRST(I)
      IPLUS=NAME(8)
      MINUS=NAME(8)
      GO TO (10,30,50,170,70,90,210),I
10    WRITE(IUNIT,21)
21    FORMAT(/1H ,11H*RESISTORS:)
      GO TO 120
30    WRITE(IUNIT,41)
41    FORMAT(/1H ,12H*CAPACITORS:)
      GO TO 120
50    WRITE(IUNIT,61)
61    FORMAT(/1H ,11H*INDUCTORS:)
      GO TO 120
70    WRITE(IUNIT,81)
81    FORMAT(/1H ,17H*CURRENT SOURCES:)
      GO TO 110
90    WRITE(IUNIT,101)
101   FORMAT(/1H ,17H*VOLTAGE SOURCES:)
110   IPLUS=NAME(9)
      MINUS=NAME(10)
120   WRITE(IUNIT,131)IPLUS,MINUS
131   FORMAT(1H ,5H*NAME,1X,A1,5HNODES,A1,4X,5HVALUE,3X,5HMODEL)
      DO 150 J=1,K1
      MPOS=KPOS/2
      WRITE(IUNIT,141)NAME(I),IELM(KPOS+2),IELM(KPOS+7),IELM(KPOS+8),
     & RELM(MPOS+3),IELM(KPOS+3)
141   FORMAT(1H ,A1,A2,2I4,G13.3,2X,1HM,I1)
      KPOS=IELM(KPOS+1)
150   CONTINUE
      GO TO 300
170   WRITE(IUNIT,181)
181   FORMAT(/1H ,13H*TRANSISTORS:/1H ,25H*NAME C     B     E    MODEL)
      DO 200 J=1,K1
      WRITE(IUNIT,191)NAME(I),IELM(KPOS+2),IELM(KPOS+7),IELM(KPOS+8),
     & IELM(KPOS+5),IELM(KPOS+3)
      KPOS=IELM(KPOS+1)
191   FORMAT(1H ,A1,A2,3I4,4X,1HM,I1)
200   CONTINUE
      GO TO 300
210   WRITE(IUNIT,221)
221   FORMAT(/1H ,8H*MODELS:/1H ,10H*NAME TYPE)
      DO 300 J=1,K1
      MPOS=KPOS/2
      K=2*RELM(MPOS+3)+3
      KK=MPOS+4
      LL=KK+7
      WRITE(IUNIT,291)NAME(7),IELM(KPOS+2),ITYPE(K),ITYPE(K+1),
     & (RELM(JJ),JJ=KK,LL)
      KPOS=IELM(KPOS+1)
291   FORMAT(1H ,A1,A2,2X,2A2,7G10.3)
300   CONTINUE
      RETURN
      END
      SUBROUTINE ALTER
C .... FIND ALTER ELEMENT
      INTEGER*2 IELM,KHALF
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
```

APPENDIX E

```
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1)),(K,KHALF)
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
C .... ITT CONTAINS FIRST LOCATION OF ALTERED ELEMENT IN IELM
      IERR=1
      K1=IELN(IEL)
      ITT=IFRST(IEL)
      K=IPACK(IAQ,LL+1)
      DO 3020 I=1,K1
      IF(IELM(ITT+2).EQ.KHALF)GO TO 3050
      ITT=IELM(ITT+1)
3020  CONTINUE
      WRITE(1W,3031)
3031  FORMAT(22H ELEMENT NOT FOUND RE-)
      IERR=0
      GO TO 3070
3050  LL=LL+3
      IF(IEL.EQ.7) GO TO 3070
      CALL RDFLD
3070  RETURN
      END
      FUNCTION VAL(LOC1)
C .... DETERMINE VALUE OF FIELD
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION SUF1X(5),ICHAR(22)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      DATA ICHAR/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,
     1 1H .,1H.,1HE,1H-,1H+,1H,,1HM,1HP,1HN,1HU,1HK,1HG/
      DATA SUF1X/1.E-12,1.E-9,1.E-6,1.E3,1.E9/
C
C                VALUE OF MM (RETURNED)
C       -1  ILLEGAL CHARACTER    1  MODEL
C        0  BLANK FIELD          2  VALID FIELD
      SIGN=1.0
      MP=1
      IS=0
      IC=0
      IINT=0
      IEXP=0
      FRAC=0.
      EMULT=1.0
      NBLNK=0
      MM=0
      J1=0
      VAL=0.0
      DO 140 LL=LOC1,80
      II=IAQ(LL)
C .... DETERMINE CHARACTER
      DO 20 J=1,22
      IF (II.EQ.ICHAR(J)) GO TO 30
20    CONTINUE
      GO TO 130
30    N=1
      J=J-1
      IF(J.LE.9) GO TO 40
      IF(J.GT.16)GO TO 136
```

```
      N=J-8
40    GO TO (50,135,90,100,110,140,150,125,130),N
50    J1=1
      IF(IS)60,70,80
C .... EXPONENT PART
60    IEXP=IEXP*10+J
      GO TO 140
C .... INTEGER PART
70    IINT=IINT*10+J
      GO TO 140
C .... FRACTION PART
80    IC=IC+1
      S=J
      FRAC=FRAC+S/PW10(IC)
      GO TO 140
C .... DECIMAL POINT
90    IS=1
      GO TO 140
C .... E
100   IF(EMULT.EQ.1.0)GO TO 105
      EMULT=1.0E6
      GO TO 140
105   IS=-1
      GO TO 140
C .... MINUS
110   IF(J1.NE.0)GO TO 115
      SIGN=-1.0
      GO TO 140
115   MP=-1
      GO TO 140
C .... MODEL
125   IF(J1.GT.0)GO TO 126
      MM=1
      GO TO 140
126   EMULT=1.E-3
      GO TO 140
C .... ERROR
130   WRITE(IW,131) II
131   FORMAT(19H ILLEGAL CHARACTER-,A2)
      MM=-1
      GO TO 180
C .... ALLOW FORM IE XX (OR IEXX OR IE+XX)
135   IF(IS.LT.0.AND.IEXP.EQ.0) GO TO 140
      IF(J1.GT.0) GO TO 150
C .... COUNT LEADING BLANKS
      IF(NBLNK.GT.7) GO TO 138
      NBLNK=NBLNK+1
      GO TO 140
136   IF(J1.EQ.0)GO TO 130
      NN=J-16
      EMULT=SUFIX(NN)
      GO TO 140
138   MM=0
      GO TO 180
140   CONTINUE
      WRITE(IW,141)
141   FORMAT(30H MAXIMUM FIELD LENGTH EXCEEDED)
      GO TO 180
150   J=MP*IEXP
      VAL=IINT
      VAL= SIGN*(VAL+FRAC)*PW10(J)*EMULT
      IF(MM.NE.1)MM=2
180   RETURN
      END
      FUNCTION PW10(K)
C .... GENERATE POWER OF TEN
      PW10=1.0
      IF(K) 15,30,5
5     DO 10 I=1,K
10    PW10=PW10*10.0
      RETURN
15    K=-K
      DO 20 I=1,K
20    PW10=PW10/10.0
30    RETURN
      END
      FUNCTION NCONV(K,M,NI,NODE)
C .... DETERMINE ELEMENT NODE FROM TABLE
      DIMENSION N1(30,2)
```

APPENDIX E

```
C     M=0  CONVERT ORIG. NODE TO RENUMBERED NODE
C     M=1  CONVERT RENUMBERED. NODE TO ORIG. NODE
      IF(M.EQ.1) GO TO 300
      DO 100 J=1,NODE
      IF(K.EQ.NI(J,2)) GO TO 200
100   CONTINUE
      GO TO 400
200   NCONV=J
      GO TO 400
300   NCONV=NI(K,2)
400   RETURN
      END
      SUBROUTINE POUT
C .... SET UP PRINT OR PLOT OUTPUTS
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IBQ(11)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      DATA IBQ/1HR,1HC,1HL,1HQ,1HI,1HV,1HM,1HE,1HT,1HP,1H /
      DATA IDB/2HDB/
C
      IFORM=0
      IF(IFLG.LT.3) GO TO 350
      WRITE(IUNIT,141)
141   FORMAT(9H OUTPUTS:/32H VXX PRT/PLO XMIN XMAX VMIN VMAX)
      READ(IR,151) IAQ
151   FORMAT(80A1)
      IOUT=VAL(2)
      I=LL+1
      LL=8
      CALL RDFLD
      IF(MM.EQ.-1) GO TO 400
      IF(IFLG.EQ.2) IFLG=6
      IF(IAQ(I).EQ.IBQ(11)) I=I+1
      IF(IAQ(I).NE.IBQ(10)) GO TO 340
      I=I+1
      IF(IAQ(I).NE.IBQ(3)) GO TO 350
C .... PLOT DEFAULT: USE LAST PLOT SCALES IF PRT/PLO NOT SPECIFIED
C ....  PLOT OUTPUT
200   KPOS=MXLOC/2
      DO 290 I=1,6
      K=KPOS+I
      RELM(K)=A(I)
290   CONTINUE
C
C .... AC PLOTS
      IF(IFLG.NE.9) GO TO 310
      LTYPE=2
      IF(A(1).EQ.0.) A(1)=TM(1)
      IF(A(2).EQ.0.) A(2)=TM(2)
      RELM(KPOS+1)=ALOG(A(1))*0.434294
      RELM(KPOS+2)=ALOG(A(2))*0.434294
C ....  ITT=1 OUTPUT DB GAIN.  ITT=2 , OUTPUT PHASE.
      ITT=TM(4)
      GO TO 340
C
C .... TRANSIENT PLOT DEFAULT
310   IF(IFLG.NE.5) GO TO 320
      LTYPE=4
      IF(A(2).EQ.0.) RELM(KPOS+2)=TM(2)
      IF(A(4).EQ.0.) RELM(KPOS+4)=20.
      GO TO 340
C
C .... SWEPT ELEMENT PLOT DEFAULT
320   IF(A(1).EQ.0.) RELM(KPOS+1)=TM(1)
      IF(A(2).EQ.0.) RELM(KPOS+2)=TM(2)
      IF(A(4).EQ.0.) RELM(KPOS+4)=20.
```

```
C
340   IPLT=1
      CALL GRAPH
      GO TO 400
C ....  PRINT OUTPUT
350   IPLT=0
C ...  DETERMINE OUTPUT PRINT DEVICE
      WRITE(IW,351)
351   FORMAT(26H OUTPUT TO: TTY(0) DISC(1))
      READ(IR,353) IFORM
353   FORMAT(I10)
      IF(IFORM.EQ.0) GO TO 366
360   CALL OPNFL(IDISC,IW,IR)
      IUNIT=IDISC
C ....  AC PRINT
366   IF(IFLG.NE.9)GO TO 370
      IAQ(1)=IBQ(7)
      IF(ITT.EQ.1) IAQ(1)=IDB
      WRITE(IUNIT,369)IAQ(1),IOUT,IOUT
369   FORMAT(3X,9HFREQUENCY,5X,1HV,A1,I2,5X,2HVP,I2,5H(DEG)/40(1H-))
      GO TO 400
C ....  ALTER PRINT
370   IF(IFLG.NE.6) GO TO 375
      IAQ(1)=IBQ(IEL)
      IAQ(2)=IELM(ITT+2)
      GO TO 380
C ....  TRANSIENT PRINT
375   IF(IFLG.NE.5) GO TO 400
      IAQ(1)=IBQ(9)
      IAQ(2)=IBQ(11)
C
380   WRITE(IUNIT,391) IAQ(1),IAQ(2),IOUT
391   FORMAT(13X,2A1,15X,1HV,I2/9X,30(1H+))
400   RETURN
      END
      SUBROUTINE SETUP
C .... PROCESS CIRCUIT DATA
      INTEGER*2 IELM
      INTEGER V1
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION NSORC(1),KCON(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(6),V1),(IAQ(1),NSORC(1)),(NI(1,2),KCON(1))
C
C .... INITIALIZE NODE VECTOR
C .... REORDER NODE VECTOR
      N=NODE-1
      DO 3200 I=1,N
      II=I+1
      DO 3200 J=II,NODE
      IF(KCON(I).LT.KCON(J))GO TO 3200
      M=KCON(J)
      KCON(J)=KCON(I)
      KCON(I)=M
3200  CONTINUE
      IF(V1.EQ.0) GO TO 3440
C .... RENUMBER VOLTAGE SOURCE NODES
      KPOS=IFRST(6)
      DO 3300 I=1,V1
      K=IELM(KPOS+7)
      NSORC(I)=NCONV(K,0,NI,NODE)
      KPOS=IELM(KPOS+1)
3300  CONTINUE
C
C .... MOVE SOURCE NODES TO END OF NODE VECTOR
      N=NODE
      KPOS=IFRST(6)
```

```
      DO 3440 I=1,VI
      K=NSORC(I)
      IF(K.EQ.N) GO TO 3430
      IF(K.GT.NNODE) GO TO 3435
      II=I+1
3350  DO 3400 L=II,VI
      IF(NSORC(L).NE.N) GO TO 3400
      N=N-1
      GO TO 3350
3400  CONTINUE
      M=KCON(K)
      KCON(K)=KCON(N)
      KCON(N)=M
3430  N=N-1
3435  KPOS=IELM(KPOS+1)
3440  CONTINUE
C .... RENUMBER ELEMENT NODES
      CALL RENUM(0)
C
C .... GENERATE INCIDENCE MATRIX FOR NODE REORDERING
      CALL INDMT
C .... DETERMINE NEW NODE ORDER AND SET UP SPARSE POINTERS
      CALL NORDR
C
C .... REDUCE VOLTAGE SOURCES TO CURRENT EQUIVALENTS
      IF(V1.EQ.0) GO TO 3920
      CALL EQUIV
      MXLOC=LPOS
3920  RETURN
      END
      SUBROUTINE INDMT
C .... ROUTINE TO LOAD INCIDENCE MATRIX FOR NODE REORDERING
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IY(30,30)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLC,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(30),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(Y(1),IY(1,1))
C
C
C ....  CLEAR IY MATRIX
      DO 100 I=1,NNODE
      DO 100 J=I,NNODE
      IY(I,J)=0
100   CONTINUE
C
C ....  LOAD INCIDENCE MATRIX
      DO 300 IEL=1,5
      KI=IELN(IEL)
      IF(KI.EQ.0) GO TO 300
      KPOS=IFRST(IEL)
      DO 200 J=1,KI
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      IF(KK.EQ.0) GO TO 110
      IY(KK,KK)=1
      IF(LL.EQ.0) GO TO 120
      IY(KK,LL)=1
      IY(LL,KK)=1
110   IY(LL,LL)=1
120   IF(IEL.NE.4) GO TO 180
C .... ADD BJT'S
      MM=IELM(KPOS+5)
      IF(MM.EQ.0) GO TO 180
      IF(KK.EQ.0) GO TO 130
      IY(KK,MM)=1
      IY(MM,KK)=1
130   IF(LL.EQ.0) GO TO 140
```

```
      IY(MM,LL)=1
      IY(LL,MM)=1
140   IY(MM,MM)=1
180   KPOS=IELM(KPOS+1)
200   CONTINUE
300   CONTINUE
      RETURN
      END
      SUBROUTINE NORDR
C .... ROUTINE TO OPTIMALLY ORDER NODES USING
C       NON-ZERO OFF-DIAGONAL TERMS (REF. BIAS-N)
C       5-18-77
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60);Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1),IROW(1),IY(30,30)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1)),(V(1),IROW(1)),(Y(1),IY(1,1))
C
      NM1=NNODE-1
      NPS=0
      NCT=0
      IF(NNODE.EQ.1) GO TO 30
      DO 10 I=1,NODE
      IORDR(I)=I
10    CONTINUE
      KNT=0
C .... COUNT NUMBER OF OFF-DIAGONAL NON-ZERO ELEMENTS IN ROWS
      DO 20 I=1,NNODE
      NCON(I)=I
      IROW(I)=0
      DO 15 J=1,NNODE
      IF(I.EQ.J) GO TO 15
      IF(IY(I,J).EQ.0) GO TO 15
      IROW(I)=IROW(I)+I
15    CONTINUE
20    CONTINUE
C
C .... COLUMN AND ROW RENUMBERING AND INDICATOR SETUP
30    IU=1
C       IL=1
      IFILL=0
      IF(NNODE.LE.1) GO TO 195
      DO 190 I=1,NM1
      IUR(I)=IU
C       ILC(I)=IL
      L=IORDR(I)
C
C .... SEACH FOR MIN IROW
      NMIN=500
      DO 120 J=I,NNODE
      NR=IORDR(J)
      IF(IROW(NR).GE.NMIN) GO TO 120
      NMIN=IROW(NR)
      IORDR(J)=L
      NCON(L)=J
      IORDR(I)=NR
      NCON(NR)=I
      L=NR
120   CONTINUE
C
C .... ESTABLISH NON-ZERO TERMS IN ROW
      JS=I+1
      DO 140 K=JS,NNODE
      IC=IORDR(K)
      IF(IY(L,IC).EQ.0)GO TO 140
      IUC(IU)=IC
      IU=IU+1
```

```
 140   CONTINUE
C
C .... MOVE DOWN COLUMN AND CHECK FILL-INS
       DO 185 J=JS,NNODE
       NR=IORDR(J)
       IRT=IY(NR,L)
       IF(IRT.EQ.0) GO TO 185
C       ILR(IL)=NR
       IROW(NR)=IROW(NR)-I
C       IL=IL+I
       NCT=IUR(I)
 145   IF(NCT.GE.IU) GO TO 185
C .... MOVE INDEX ,IC, ACROSS ROW L
 150   IC=IUC(NCT)
       IF(NR.EQ.IC) GO TO 180
       IF(IY(NR,IC).NE.0) GO TO 180
C .... OFF-DIAGONAL FILL-IN
       IROW(NR)=IROW(NR)+I
       IFILL=IFILL+1
       IY(NR,IC)=I
 180   NCT=NCT+1
       NPS=NPS+I
       GO TO 145
 185   CONTINUE
 190   CONTINUE
C       ILC(NNODE)=IL
 195   IUR(NNODE)=IU
       MLOC=NODE+IU
       MXPOS=LPOS
C
C .... PRINT MATRIX STATISTICS
       IUT=IU-I
       NOPS=NPS+NNODE+3*IUT
       I=NNODE*NNODE
       J=NODE+2*IU
       NCT=100.*FLOAT(I-2*IUT)/FLOAT(I)
       WRITE(IUNIT,201) NNODE,IUT,NOPS,IFILL,NCT,MXPOS,J
 201   FORMAT(4X,7HNNODE =,I4,7X,4HIU =,I4/
      8 3X,6HNOPS =,I4,2X,9HFILL-INS=,
      8 I4/1X,10HSPARSITY =,I4,2H %,3X,6HMXPOS=,I4,3X,7HMXYPOS=,I4)
       J=NODE+2*IU
       IF(J.GT.300) WRITE(IW,211)
 211   FORMAT(23H ** MATRIX TOO DENSE **)
C
C .... ASSIGN OPERATION NUMBERS
       IF(NNODE.EQ.1) GO TO 230
       NPS=0
       DO 225 I=1,NM1
       IUS=IUR(I)
       IUE=IUR(I+1)
       IL=IUS
       ILE=IUE
 205   IF(IL.GE.ILE) GO TO 225
       NR=ILR(IL)
       IL=IL+I
       IU=IUS
 215   IF(IU.GE.IUE) GO TO 205
       NC=IUC(IU)
       NPS=NPS+I
       IPOS(NPS)=INDX(NR,NC)
       IU=IU+1
       GO TO 215
 225   CONTINUE
       IF(NPS.GT.400) WRITE(IW,211)
 230   RETURN
       END
       SUBROUTINE RENUM(M)
C .... RENUMBER ELEMENT NODES
       INTEGER*2 IELM
       DOUBLE PRECISION V(60),U(30),C(60),Y(600)
       DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
       DIMENSION ILC(1),ILR(1),RELM(1)
       COMMON U,C,Y,DS,DELT,DELTA
       COMMON T0,TEMP,DTEMP
       COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
       COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
       COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
       COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
       COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
```

```
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      DO 4000 IEL=1,6
      K1=IELN(IEL)
      IF(K1.EQ.0)GO TO 4000
      KPOS=IFRST(IEL)
      DO 4000 J=1,K1
      KK=7
      IF(IEL.EQ.4) KK=5
      DO 3000 I=KK,8
      IF(I.EQ.6) GO TO 3000
      II=KPOS+I
      K=IELM(II)
      IF(K.EQ.0) GO TO 3000
      IELM(II)=NCONV(K,M,NI,NODE)
3000  CONTINUE
3600  KPOS=IELM(KPOS+1)
4000  CONTINUE
      RETURN
      END
      SUBROUTINE EQUIV
C .... STORE LOCATION OF EQUIVALENT SOURCES
      INTEGER*2 IELM
      INTEGER V1
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(IELN(6),V1),(IELN(9),I2)
C
      KPOS=IFRST(6)
      LPOS=MXPOS
      DO 1500 N=1,V1
      J=IELM(KPOS+7)
C .... CHECK IF ELEMENT CONNECTED TO VOLTAGE SOURCE
      DO 1400 IEL=1,3
      K1=IELN(IEL)
      IF(K1.EQ.0) GO TO 1300
      MPOS=IFRST(IEL)
      DO 1400 M=1,K1
 K=IELM(MPOS+7)
      L=IELM(MPOS+8)
      IF(J.NE.K)GO TO 1000
      NT=L
      GO TO 1100
1000  IF(L.NE.J)GO TO 1300
      NT=K
1100  IF(NT.EQ.0)GO TO 1300
      I2=I2+1
      IF(I2.EQ.1) IFRST(9)=MXPOS
      IF(LPOS.LE.MXLST) GO TO 1200
      IERR=-2
C .... STORE EQUIVALENT SOURCE FLAGS AND VALUES
C       LPOS+1= ELEMENT LOCATION IN IELN(.)
C       LPOS+2= VOLTAGE SOURCE LOCATION IN IELN(.)
C       LPOS+3= NODE AT WHICH EQUIVALENT CURRENT IS ADDED
C       LPOS+4= ELEMENT TYPE
1200  IELM(LPOS+4)=IEL
      IELM(LPOS+3)=NT
      IELM(LPOS+2)=KPOS/2
      IELM(LPOS+1)=MPOS/2
      LPOS=LPOS+4
1300  MPOS=IELM(MPOS+1)
1400  CONTINUE
      KPOS=IELM(KPOS+1)
1500  CONTINUE
```

APPENDIX E

```
      RETURN
      END
      SUBROUTINE GRAPH
C .... INITIALIZE GRAPHICS
C .... DRAW AXIS AND LABEL GRAPH
      RETURN
      END
      FUNCTION IPACK(IAQ,K)
C .... PACK 2A1 FORMAT INTO II WORD
      DIMENSION IAQ(1)
      II=IAQ(K)
      JJ=IAQ(K+1)
C       IPACK=OR(SHFT(II,8,-8),SHFT(JJ,8))
      IPACK=II
      RETURN
      END
      SUBROUTINE OPNFL(LUN,IW,IR)
      DIMENSION NAME(3)
C .... ROUTINE TO OPEN DISCFILE
      WRITE(IW,101)
101   FORMAT(10H FILENAME:)
      READ(IR,201)NAME
201   FORMAT(3A2)
      CALL SEARCH(LUN,NAME,1)
      RETURN
      END
      SUBROUTINE CLSFL(LUN)
C .... ROUTINE TO CLOSE DISCFILE
      CALL SEARCH(4,0,1)
      RETURN
      END
      SUBROUTINE SECND(TI)
C .... ROUTINE TO RETURN ELAPSED TIME
      DIMENSION ITAR(11)
      CALL TIMDAT(ITAR,11)
      TI=FLOAT(ITAR(7))+FLOAT(ITAR(8))/FLOAT(ITAR(11))
      RETURN
      END
      SUBROUTINE CLOCK(ETIM,IDATE)
C .... RETURNS CLOCK TIME IN SECONDS, AND DATE (MM DD Y)
      DIMENSION IDATE(3)
      IDATE(1)=0
      IDATE(2)=0
      IDATE(3)=0
      ETIM=0.
      RETURN
      END
       SUBROUTINE ANALY
C .... MAIN ANALYSIS ROUTINE
      INTEGER*2 IELM
      INTEGER R1,C1,VI,QI
      DOUBLE PRECISION DELU
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(IELN(1),R1),(IELN(2),C1),(IELN(3),L1),(IELN(4),QI),
     1 (IELN(5),II),(IELN(6),VI),(IELN(7),MI),(IELN(9),I2)
C
      ERR=FLOAT(NNODE*NNODE)*1.0E-10
2900  RMS1=0.
      RMS2=0.
      IF(IFLG.NE.5)GO TO 3200
3000  IF(T0.NE.DELTA)GO TO 3100
      DELT=DELTA
C .... UPDATE TRANSIENT SOURCES
3100  CALL UPDAT
C .... UPDATE CAPACITOR CURRENTS
3200  IF(C1.LE.0) GO TO 3800
```

```
      KPOS=IFRST(2)
      TC=1.0
      IF(ITEMP.EQ.0)GO TO 3300
      TC=1.0+RELM(ITEMP+6)*DTEMP+RELM(ITEMP+7)*DTEMP*DTEMP
3300  DO 3700 I=1,C1
      MPOS=KPOS/2
      DS=0.D0
      IF(T0.LE.0.)GO TO 3400
      DS=RELM(MPOS+3)*TC
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      DS=2.D0*DS*DELU(LL,KK)/DELT
      IF(T0.GT.DELTA)GO TO 3500
3400  RELM(MPOS+5)=0.D0
      RELM(MPOS+6)=-DS
      GO TO 3600
3500  RELM(MPOS+5)=DS+RELM(MPOS+6)
      RELM(MPOS+6)=-DS-RELM(MPOS+5)
3600  KPOS=IELM(KPOS+1)
3700  CONTINUE
C .... UPDATE INDUCTOR CURRENTS
3800  IF(L1.EQ.0) GO TO 4200
      TC=1.0
      KPOS=IFRST(3)
      DO 4100 I=1,L1
      MPOS=KPOS/2
      IF(T0.LE.0.)GO TO 3900
      DS=RELM(MPOS+3)*TC
      DS=DELT*DELU(LL,KK)/DS/2.D0
      IF (T0.GT.DELTA) GO TO 4000
3900  RELM(MPOS+5)=0.D0
      RELM(MPOS+6)=DELU(LL,KK)*100.D0
      GO TO 4100
4000  RELM(MPOS+5)=DS+RELM(MPOS+6)
      RELM(MPOS+6)=RELM(MPOS+5)+DS
      KPOS=IELM(KPOS+1)
4100  CONTINUE
C .... ADD SUPPLIES TO VOLTAGE VECTOR
4200  IF(V1.EQ.0) GO TO 4400
      KPOS=IFRST(6)
      DO 4300 I=1,V1
      MPOS=KPOS/2
      J=IELM(KPOS+7)
      V(J)=RELM(MPOS+3)
      U(J)=V(J)
      KPOS=IELM(KPOS+1)
4300  CONTINUE
4400  ITER=0
C .... ZERO CURRENT MATRIX
4500  DO 4600 I=1,NNODE
      C(I)=0.D0
4600  CONTINUE
      II=2*MLOC-NODE
      DO 4700 J=1,II
      Y(J)=0.D0
4700  CONTINUE
C .... LOAD ELEMENTS INTO Y & C ARRAYS
4800  CALL ELOAD
      IF(I2.EQ.0) GO TO 5520
C .... ADD GENERATED CURRENT SOURCES
      CALL GNCUR
C .... SOLVE NODE EQUATIONS
5520  CALL DECMP
      CALL SOLVE
      IF(Q1.EQ.0) GO TO 6550
      ITER=ITER+1
      IF(ITER.LT.100)GO TO 6500
C .... NO CONVERGENCE--PRINT LAST NODE VOLTAGES
      WRITE(IUNIT,6501)
6501  FORMAT(26H CIRCUIT DOES NOT CONVERGE)
      GO TO 6780
C .... COMPUTE MEAN SQUARE ERROR OF NODE VOLTAGE CHANGES
6500  DS=0.D0
      DO 6520 J=1,NNODE
      DS=DS+(V(J)-U(J))**2
6520  CONTINUE
      S=DS
      IF(IFLG.EQ.5.OR.IFLG.EQ.6)GO TO 6550
      WRITE(IUNIT,6541)S
```

APPENDIX E

```
6541  FORMAT(1X,E18.4)
C .... STORE LAST NODE VOLTAGES
6550  DO 6560 I=1,NNODE
      U(I)=V(I)
6560  CONTINUE
      IF(Q1.EQ.0) GO TO 6660
C .... CHECK FOR CONVERGENCE
      IF(S.LT.ERR.AND.RMS1.LT.ERR.AND.RMS2.LT.ERR.AND.ITER.GT.2)
     8 GO TO 6660
      IF(ITER.LT.6) GO TO 6580
      IF(S.GE.RMS1.AND.RMS1.GE.RMS2.AND.S.LT.0.001) GO TO 6600
6580  RMS2=RMS1
      RMS1=S
C .... NO CONVERGENCE: RE-ITERATE
      GO TO 4500
6600  IF(IPLT.GT.0)GO TO 6660
      S=SQRT(S)
      WRITE(IUNIT,6601)S
6601   FORMAT(19H MEAN ERROR(VOLTS):,F14.6)
6660  IF(IFLG.NE.5.AND.IFLG.NE.6)GO TO 6770
      ITOTL=ITOTL+ITER
      R=T0
      S=V(JOUT)
      IF(IFLG.EQ.6) R=TM(1)
      IF(IPLT.EQ.0)GO TO 6670
C      CALL DRAW(R,S,IPEN,4)
      IPEN=1
      GO TO 6680
6670  WRITE(IUNIT,6671) R,S,ITER
6671  FORMAT(1X,2G18.4,10X,I10)
6680  IF(IFLG.EQ.6)GO TO 6700
      T0=T0+DELTA
      IF(T0.GT.TM(2)) GO TO 6965
      GO TO 3000
C
C .... INCREMENT SWEPT ALTER VALUE
6700  TM(1)=TM(1)+TM(3)
      MPOS=TM(4)
      RELM(MPOS+3)=TM(1)
      IF(TM(1).LE.TM(2)) GO TO 2900
      RELM(MPOS+3)=TM(5)
      GO TO 6965
C
C .... PRINT DC OPERATING POINTS
6770  IF(Q1.EQ.0)GO TO 6800
6780  WRITE(IUNIT,6781)ITER
6781  FORMAT(//12H ITERATIONS:,I10)
6800  TC=TEMP-273.
      WRITE(IUNIT,6811)TC
6811  FORMAT(3H T=,F8.1,6H DEG C//)
      WRITE(IUNIT,6831)
6831  FORMAT( 15H NODE VOLTAGES:)
      DO 6890 I=1,NODE
      J=NI(I,2)
      WRITE (IUNIT,6881)J,V(I)
6881  FORMAT(2H V,I2,F18.4)
6890  CONTINUE
      IFLG=4
      IF(Q1.NE.0) CALL ELOAD
C
6965  IFLG=4
6970  RETURN
      END
      SUBROUTINE ELOAD
C .... ROUTINE TO LOAD ELEMENTS INTO Y & C ARRAYS
C          FOR AC OR DC ANALYSES
      INTEGER*2 IELM
      INTEGER Q1
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
```

```
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(4),Q1),(IELN(5),II)
C
      DO 800 IEL=1,3
      K1=IELN(IEL)
      IF(K1.EQ.0) GO TO 800
      KPOS=IFRST(IEL)
      TC=1.0
      IF(ITEMP.EQ.0) GO TO 100
      NN=ITEMP+IEL*2+1
      TC=1.0+RELM(NN)*DTEMP+RELM(NN+1)*DTEMP**2
100   DO 800 I=1,K1
      IF(IELM(KPOS+4).NE.0) CALL ELMOD(IEL)
      MPOS=KPOS/2
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      DS=RELM(MPOS+3)*TC
C .... ADD RESISTORS
      IF(IEL.GT.1) GO TO 200
      DS=1.D0/DS
      CALL ADRES
      GO TO 700
C .... ADD CAPACITORS
200   IF(IEL.GT.2) GO TO 400
      DS=DS/DELT
      IF(IFLG.NE.9) GO TO 300
      CALL ADCPR
      GO TO 700
300   DS=2.D0*DS
      GO TO 600
C .... ADD INDUCTORS
400   DS=DELT/DS
      IF(IFLG.NE.9) GO TO 500
      CALL ADCPR
      GO TO 700
500   DS=DS*0.5D0
      IF(DELT.GT.1.0D6) DS=100.D0
600   CALL ADRES
      DS=RELM(MPOS+6)
      CALL ADCUR
700   KPOS=IELM(KPOS+1)
800   CONTINUE
      IF(II.EQ.0) GO TO 1000
C
C .... ADD CURRENT SOURCES FOR DC ANALYSIS
      IF(IFLG.EQ.9) GO TO 1000
      KPOS=IFRST(5)
      DO 900 I=1,II
      MPOS=KPOS/2
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      DS=RELM(MPOS+3)
      CALL ADCUR
      KPOS=IELM(KPOS+1)
900   CONTINUE
C .... ADD TRANSISTORS
1000  IF(Q1.EQ.0) GO TO 1200
      IF(IFLG.EQ.9) GO TO 1100
      CALL BJT
      GO TO 1200
1100  CALL BJTAC
1200  RETURN
      END
      SUBROUTINE DECMP
C .... PERFORMS LU DECOMPOSITION BASED ON RECORDED SPARSITY
C      MODIFIED FROM SINC-S 5-18-77
C
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
```

```
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1))
C
C     IUR  UPPER TRIANGULAR ROW ELEMENT COUNTER
C     IUC  UPPER TRIANGULAR ELEMENT COLUMN INDICATOR
C     ILC  LOWER TRIANGULAR COLUMN ELEMENT COUNTER
C     ILR  LOWER TRIANGULAR ELEMENT ROW INDICATOR
C
      IF(NNODE.EQ.1) GO TO 40
      NN=NNODE-1
      KNT=0
      DO 30 I=1,NN
      L=IORDR(I)
      IUS=IUR(I)+MLOC
      IUE=IUR(I+1)+MLOC
      IL=ILC(I)+NODE
      ILE=ILC(I+1)+NODE
C
C .... DOWN LOWER TRIANGLE COLUMNS
5     IF(IL.GE.ILE) GO TO 30
      DS=Y(IL)/Y(L)
      Y(IL)=DS
      IL=IL+1
      IU=IUS
C
C ....  ACROSS UPPER TRIANGLE ROWS
20    IF(IU.GE.IUE) GO TO 5
      KNT=KNT+1
      K=IPOS(KNT)
      Y(K)=Y(K)-Y(IU)*DS
      IU=IU+1
      GO TO 20
30    CONTINUE
40    RETURN
      END
      SUBROUTINE SOLVE
C
C .... PERFORMS FORWARD AND BACKWARD SUBSTITUTION
C       USING SPARSE POINTERS
C           FROM BIAS-N 5-19-77
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1))
C
C .... FORWARD SUBSTITUTION
      NN=NNODE-1
      IF(NN.GT.0) GO TO 10
      V(1)=C(1)/Y(1)
      GO TO 70
10    DO 30 I=1,NN
      L=IORDR(I)
      DS=C(L)
      IF(DS.EQ.0.D0) GO TO 30
      IL=ILC(I)
      ILE=ILC(I+1)
20    NL=IL+NODE
      IF(IL.GE.ILE) GO TO 30
      NR=ILR(IL)
      C(NR)=C(NR)-Y(NL)*DS
      IL=IL+1
```

```
      GO TO 20
30    CONTINUE
C
C .... BACK SUBSTUTION
      L=IORDR(NNODE)
      C(L)=C(L)/Y(L)
      DO 50 I=I,NN
      NUI=NNODE-I
      L=IORDR(NUI)
      IU=IUR(NUI)
      IUE=IUR(NUI+I)
35    NL=IU+MLOC
      IF(IU.GE.IUE) GO TO 45
40    IC=IUC(IU)
      C(L)=C(L)-Y(NL)*C(IC)
      IU=IU+I
      GO TO 35
45    C(L)=C(L)/Y(L)
50    CONTINUE
C .... TRANSFER INTO VOLTAGE VECTOR
C       DO 60 I=I,NNODE
C       V(I)=C(I)
C60   CONTINUE
70    RETURN
      END
      SUBROUTINE ELMOD(IEL)
C .... ROUTINE TO DEFINE ELEMENT MODELS
      GO TO (200,400,600),IEL
C .... RESISTOR MODEL
200   RETURN
C .... CAPACITOR MODEL
400   RETURN
C .... INDUCTOR MODEL
600   RETURN
      END
      SUBROUTINE BJT
C .... COMPUTE BJT PARAMETERS
      INTEGER Q1
      INTECER*2 IELM
      DOUBLE PRECISION DELU
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(I),CI(I),YI(I)
      DIMENSION ILC(I),ILR(1),RELM(I)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(I),ILC(I)),(IUC(1),ILR(I)),
     8 (IELM(1),RELM(I))
      EQUIVALENCE (C(1),V(I)),(C(30),CI(1),VI(1)),(Y(300),YI(I))
      EQUIVALENCE(IELN(4),Q1)
C
      IF(IFLG.NE.4) GO TO 5400
      WRITE(IUNIT,5201)
5201  FORMAT(29H TRANSISTOR OPERATING POINTS:)
      WRITE (IUNIT,5301)
5301  FORMAT(5H NAME,5X,2HIB,9X,2HIC,7X,3HVBE,9X,3HVBC,6X,4HBETA,7X,
     I 2HGM,9X,3HRPI)
5400  VT=8.6164E-05*TEMP
      VCT=VT*ALOG(VT/1.4I459)
      C0=1.0
      TC=I.0
      KPOS=IFRST(4)
      IF(ITEMP.EQ.0)CO TO 5500
      C0=(TEMP/300.)**3*EXP(-13920.0*(I.0/TEMP-I,0/300.))
      TC=1.0+DTEMP*RELM(ITEMP+7)+DTEMP**2*RELM(ITEMP+8)
5500  DO 7200 I=I,Q1
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      MM=IELM(KPOS+5)
      MPOS=KPOS/2
      ITT=IELM(KPOS+4)
      TYPE=RELM(ITT+3)
      BF0=RELM(ITT+4)
      BR0=RELM(ITT+5)
```

```
      CS0=RELM(ITT+6)
C
C .... INITIALIZE PARAMETERS FOR FIRST ITERATION
      IF(ITER.NE.0) GO TO 5600
      IF(IFLG.NE.1)GO TO 5600
      VBE=VCT-VT*ALOG(C0*CS0)
      VBC=-1.0
      RELM(MPOS+5)=0.5
      RELM(MPOS+6)=0.0
      RELM(MPOS+7)=1.0E-4
      RELM(MPOS+8)=1.0E-12
      GO TO 5700
C
5600  VBE=DELU(MM,LL)*TYPE
      VBC=DELU(KK,LL)*TYPE
5700  VA=RELM(ITT+7)
      VA1=1.0
      IF(VBC.GE.0.)GO TO 5710
      IF(VBC.LT.-VA) GO TO 5710
      VA1=1.0-VBC/VA
C .... PROCESS FORWARD TRANSISTOR
5710  BF=BF0*TC*VA1
      C0=C0*VA1
      CSAT=C0*CS0*(1.0+1.0/BF0)/(1.0+1.0/BF)
      CALL JUNCT(MPOS+5,VBE,CCC,CCEQ,GMF)
      IF(RELM(ITT+8).NE.0.)GO TO 5800
      GRPI=0.0
      CREQ=0.0
      CREC=0.0
      GO TO 6000
C .... INCLUDE GENERATION RECOMBINATION CURRENT
5800  CRSAT=SQRT(RELM(ITT+8)*CSAT)/BF0
      CREC=-CRSAT
      IF(VBE.LT.-1.2)GO TO 5990
      CREC=CRSAT*EXP(VBE/VT/2.)+CREC
5990  GRPI=(CREC+CRSAT)/VT/2.
      CREQ=TYPE*(CREC-GRPI*VBE)
6000  GPIF=GMF/BF+GRPI
C .... PROCESS REVERSE TRANSISTOR
      BR=BR0*TC*VA1
      CSAT=C0*CS0*(1.0+1.0/BR0)/(1.0+1.0/BR)
      CALL JUNCT(MPOS+6,VBC,CEC,CEEQ,GMR)
      GPIR=GMR/BR
      GMR=GMR+(CCC-CEC)/VA
      IF(IFLG.NE.4)GO TO 6150
C
C .... PRINT TRANSISTOR OPERATING POINTS
      CB=TYPE*(CCC/BF+CEC/BR+CREC)
      CC=TYPE*(CCC-CEC-CEC/BR)
      VBE=TYPE*VBE
      VBC=TYPE*VBC
      BF=CC/CB
      GPIF=1.0/GPIF
      WRITE (IUNIT,6121)IELM(KPOS+2),CB,CC,VBE,VBC,BF,GMF,GPIF
6121  FORMAT(2H Q,A2,2E12.3,2F9.3,F11.2,2E12.3)
      GO TO 7100
C
C .... GND. CONDUCTANCES AND V.D.G.S. CONNECTED TO SUPPLIES
6150  IF(KK.LE.NNODE)GO TO 6160
      CEEQ=CEEQ-GMR*U(KK)
      KK=0
6160  IF(LL.LE.NNODE)GO TO 6170
      CCEQ=CCEQ+GMF*U(LL)
      CEEQ=CEEQ+GMR*U(LL)
      LL=0
6170  IF(MM.LE.NNODE)GO TO 6180
      CCEQ=CCEQ-GMF*U(MM)
      MM=0
C
C .... LOAD ADMITTANCE MATRIX
6180  IF(KK.EQ.0)GO TO 6300
      Y(KK)=Y(KK)+GMR+GPIR
      IF(LL.EQ.0)GO TO 6270
      JJ= INDX(KK,LL)
      Y(JJ)=Y(JJ)+GMF-GMR-GPIR
      Y(NN)=Y(NN)-GPIR
6270  IF(MM.EQ.0)GO TO 6360
      JJ= INDX(KK,MM)
      Y(JJ)=Y(JJ)-GMF
```

```
      Y(NN)=Y(NN)-GMR
6300  IF(MM.EQ.0)GO TO 6360
      Y(MM)=Y(MM)+CMF+CPIF
      IF(LL.EQ.9)GO TO 6390
      JJ= INDX(MM,LL)
      Y(JJ)=Y(JJ)-CMF-CPIF+GMR
      Y(NN)=Y(NN)-GPIF
6360  IF(LL.EQ.0)GO TO 6390
      Y(LL)=Y(LL)+CPIF+CPIR
C .... LOAD CURRENT VECTOR
6390  IF(IFLC.EQ.9) GO TO 7100
C .... DONT ADD DC CURRENTS IF AC ANALYSIS
      IF(KK.EQ.0)GO TO 7010
      C(KK)=C(KK)+(1.0+1.0/BR)*CEEQ-CCEQ
7010  IF(MM.EQ.0) GO TO 7020
      C(MM)=C(MM)+(1.0+1.0/BF)*CCEQ-CEEQ+CREQ
7020  IF(LL.EQ.0) GO TO 7100
      C(LL)=C(LL)-CCEQ/BF-CEEQ/BR-CREQ
7100  KPOS=IELM(KPOS+1)
7200  CONTINUE
      RETURN
      END
      SUBROUTINE JUNCT(J,VBB,CCC,CEQ,CM)
C .... DETERMINE TRANSISTOR IC,GM,CPI
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
C .... COLN LIMITING ALGORITHM IMPLEMENTED 8/16/76
      VCRIT=VCT-VT*ALOG(CSAT)
      IF(VBB.LE.VCRIT)GOTO 1100
C .... VB.LT.VCRIT-- ITERATE ON VOLTAGE
      CCC=CSAT*(EXP(RELM(J)/VT)-1.0)
      CEQ=CCC+RELM(J+2)*(VBB-RELM(J))
      IF(CEQ.LT.0.)GO TO 1000
C .... ITERATE ON CURRENT
      VBB=VT*ALOG(CEQ/CSAT+1.0)
      GO TO 1100
1000  VBB=VCRIT
1100  CCC=-CSAT
      IF(VBB.LT.-1.2)GOTO 1200
      CCC=CSAT*EXP(VBB/VT)+CCC
1200  GM=(CCC+CSAT)/VT+1.0E-10
      CEQ=TYPE*(CCC-GM*VBB)
      RELM(J+2)=GM
      RELM(J)=VBB
      RETURN
      END
      SUBROUTINE UPDAT
C .... UPDATE TRANSIENT SOURCES
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      DO 4700 IEL=5,6
```

```
      K1=IELN(IEL)
      IF(K1.EQ.0)GO TO 4700
      KPOS=IFRST(IEL)
      DO 4700 J=1,K1
      MPOS=KPOS/2
      ITT=IELM(KPOS+4)
      IF(ITT.EQ.0)GO TO 4610
      L=RELM(ITT+3)-2.0
      IF(L.EQ.2) GO TO 4590
      B1=RELM(ITT+4)
      B2=RELM(ITT+5)
      B3=RELM(ITT+6)
      B4=RELM(ITT+7)
      IF(L.EQ.3) GO TO 4410
C
C .... SINE
4370  V0=B1
      IF(B4.NE.0.)GO TO 4378
      B4=DELTA
4378  IF(T0.LT.B4)GOTO 4390
      V0=V0+B2*SIN(6.28319*B3*(T0-B4)
     8 +RELM(ITT+8)/57.296)
4390  GO TO 4600
C
C .... PULSE
4410  TI=T0
4420  Z=B3
      IF(TI.GT.Z)GOTO 4450
      V0=B1
      GO TO 4600
4450  Z=Z+B4
      IF(TI.GE.Z)GOTO 4490
      V0=B2-(RELM(ITT+5)-B1)/B4*(Z-TI)
      GO TO 4600
4490  Z=Z+RELM(ITT+8)
      IF(TI.GT.Z)GOTO 4530
      V0=B2
      GO TO 4600
4530  Z=Z+RELM(ITT+9)
      IF(TI.GE.Z)GOTO 4570
      V0=B1+(B2-B1)/RELM(ITT+9)*(Z-TI)
      GO TO 4600
4570  S=RELM(ITT+10)
      IF(S.EQ.0.)GO TO 4580
      Z=Z+S
      TI=TI-S
      GO TO 4420
4580  V0=B1
      GO TO 4600
C
4590  CALL VEXT(T0,V0,ITT,RELM)
4600  RELM(MPOS+3)=V0
4610  KPOS=IELM(KPOS+1)
4700  CONTINUE
      RETURN
      END
      SUBROUTINE VEXT(T0,V0,I,RELM)
C .... USER DEFINABLE SUBROUTINE FOR SOURCE MODEL 'EXT'
      DIMENSION RELM(1)
C
C                ***PARAMETERS***
C           T0--TIME(SEC) PASSED TO VEXT
C           V0--SOURCE VALUE AT TIME T0 ,RETURNED FROM VEXT
C           RELM(*)-- AVAILABLE PARAMETERS FROM 'EXT' MODEL FIELD
C           I -- MODEL NUMBER
      RETURN
      END
      FUNCTION DELU(K,L)
C .... DETERMINE U(L)-U(K)
      INTEGER*2 IELM
      DOUBLE PRECISION DELU
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
```

```
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,1ERR,MLOC,KPOS,LPOS
      COMMON 1AQ(80),N1(30,2),1ELN(9),1FRST(9),1LAST(9),1ELM(1000)
      COMMON 1UR(30),1UC(120),1POS(400),NCON(30)
      EQUIVALENCE (1UR(1),1LC(1)),(1UC(1),1LR(1)),
     & (1ELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      DELU=0.D0
      IF(L.GT.0) DELU=U(L)
      IF(K.GT.0)DELU=DELU-U(K)
      RETURN
      END
      SUBROUTINE ADRES
C .... ADD RES1STORS TO Y MATR1X
      1NTEGER*2 IELM
      DOUBLE PREC1SION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,V1(1),C1(1),YI(1)
      D1MENSION 1LC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON 1EL,JJ,KK,LL,MM,NN,1FLG,JFLG,1TT,1TER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,1ERR,MLOC,KPOS,LPOS
      COMMON 1AQ(80),N1(30,2),1ELN(9),1FRST(9),1LAST(9),1ELM(1000)
      COMMON IUR(30),IUC(120),1POS(400),NCON(30)
      EQUIVALENCE (1UR(1),1LC(1)),(IUC(1),1LR(1)),
     & (IELM(1),RELM(1))
      EQU1VALENCE (C(1),V(1)),(C(30),CI(1),V1(1)),(Y(300),YI(1))
C
      IF(KK.GT.NNODE)KK=0
      1F(LL.GT.NNODE)LL=0
      IF(KK.EQ.0)GOTO 5100
      Y(KK)=Y(KK)+DS
      1F(LL.EQ.0)GOTO 5200
      JJ= 1NDX(KK,LL)
      Y(JJ)=Y(JJ)-DS
      Y(NN)=Y(NN)-DS
5100  1F(LL.EQ.0)GOTO 5200
      Y(LL)=Y(LL)+DS
5200  RETURN
      END
      SUBROUTINE ADCUR
C .... ADD CURRENTS TO CURRENT VECTOR
      1NTEGER*2 IELM
      DOUBLE PREC1SION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),C1(1),YI(1)
      D1MENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,ID1SC,IUNIT
      COMMON IPLT,1PEN,LTYPE,1TEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,1ERR,MLOC,KPOS,LPOS
      COMMON 1AQ(80),N1(30,2),1ELN(9),1FRST(9),1LAST(9),1ELM(1000)
      COMMON IUR(30),IUC(120),1POS(400),NCON(30)
      EQUIVALENCE (1UR(1),1LC(1)),(1UC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      IF(KK.GT.NNODE)KK=0
      1F(LL.GT.NNODE)LL=0
      1F(KK.EQ.0)GOTO 5400
      C(KK)=C(KK)-DS
5400  1F(LL.EQ.0)GOTO 5500
      C(LL)=C(LL)+DS
5500  RETURN
      END
      SUBROUTINE GNCUR
C .... ROUTINE TO ADD GENERATED CURRENT SOURCES
      INTEGER*2 1ELM
      D0UBLE PREC1S1ON V(60),U(30),C(60),Y(600)
      DOUBLE PREC1S1ON DELTA,DELT,DS,V1(1),C1(1),YI(1)
      D1MENS1ON ILC(1),1LR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON 1EL,JJ,KK,LL,MM,NN,1FLG,JFLG,1TT,1TER,IW,1R,ID1SC,IUN1T
      COMMON 1PLT,1PEN,LTYPE,1TEMP,1TOTL,1OUT,JOUT,1NPT,1FORM
```

```
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(IELN(9),I2)
C
      LL=0
      KPOS=IFRST(9)
      DO 1400 I=1,I2
      JPOS=IELM(KPOS+1)
      NPOS=IELM(KPOS+2)
C .... DURING AC ANALYSIS ONLY ADD AC SOURCE CURRENTS
      IF(INPT.NE.NPOS.AND.IFLG.EQ.9) GO TO 1390
      KK=IELM(KPOS+3)
      M=IELM(KPOS+4)
      TC=1.0
      IF(ITEMP.EQ.0) GO TO 1350
      NN=ITEMP+2*M+1
      TC=1.0+RELM(NN)*DTEMP+RELM(NN+1)*DTEMP*DTEMP
C
C .... RESISTOR CONNECTED TO VOLTAGE SOURCE
1350  IF(M.GT.1)GO TO 1360
      DS=-RELM(NPOS+3)/(RELM(JPOS+3)*TC)
      GO TO 1375
C
C .... CAPACITOR CONNECTED TO VOLTAGE SOURCE
1360  IF(M.GT.2)GO TO 1370
1365  DS=RELM(NPOS+3)*RELM(JPOS+3)*TC
      DS=-DS/DELT
      IF(IFLG.EQ.9) GO TO 1380
      DS=DS*2.0D0
      GO TO 1375
C
C .... INDUCTOR CONNECTED TO VOLTAGE SOURCE
1370  IF(M.GT.3)GO TO 1400
      DS=RELM(NPOS+3)/RELM(JPOS+3)
      DS=-DS*DELT
      IF(IFLG.EQ.9) GO TO 1380
      DS=DS*0.5D0
1375  CALL ADCUR
      GO TO 1390
1380  CALL ADCPC
1390  KPOS=KPOS+4
1400  CONTINUE
      RETURN
      END
      SUBROUTINE ACSOL
C      MAIN AC ANALYSIS ROUTINE USED WITH BIAST8. 8-12-77
C .... USING STANDARD AC ANALYSIS PROCEEDURE WITH AC BJT LOAD
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(9),I2)
C
      FSTOP=TM(2)
      NDEC=TM(3)
      IPRT=TM(4)
      FREQ=TM(1)
      FMULT=FREQ
      IFREQ=0
C .... ZERO DC SOURCES
      DO 900 I=NNODE,NODE
      V(I)=0.D0
900   CONTINUE
      VIN=RELM(INPT+3)
```

```
C .... DETERMINE INPUT NODE
      IN=INPT*2
      IN=IELM(IN+7)
C .... STORE UNITY VOLTAGE IN AC INPUT SOURCE
      RELM(INPT+3)=1.0
      IF(IEL.EQ.6) V(IN)=1.D0
920   W=6.2831*FREQ
      DELT=1.0/W
C .... ZERO COMPLEX CURRENT VECTOR
      DO 1050 I=1,NODE
      C(I)=0.D0
      CI(I)=0.D0
1050  CONTINUE
C .... ZERO COMPLEX ADMITTANCE MATRIX
      II=2*MLOC-NODE
      DO 1100 I=1,II
      Y(I)=0.D0
      YI(I)=0.D0
1100  CONTINUE
C .... LOAD AC MATRIX
      CALL ELOAD
C .... IF CURRENT SOURCE INPUT
      IF(IEL.NE.5) GO TO 1200
      II=INPT*2+7
      KK=IELM(II)
      LL=IELM(II+1)
      DS=1.0D0
      CALL ADCUR
C .... ADD GENERATED CURRENT SOURCES
1200  IF(I2.EQ.0) GO TO 1250
      CALL GNCUR
C .... FORWARD AND BACK SUBSTITUTE TO GET NEW VOLTAGES
1250  CALL DECAC
      CALL SOLAC
C
      U1=V(JOUT)
      U2=VI(JOUT)
      VMAG=U1*U1+U2*U2
1500  AMAG=SQRT(VMAG)
      IF(IPLT.EQ.0) GO TO 1600
      IF(IPRT.NE.2) GO TO 1900
1600  PHASE=ATAN(U2/U1)*57.2958
      IF(U1.GT.0.) GO TO 1900
      PHASE=PHASE+SIGN(180.,U2)
C
1900  IF(IPRT.EQ.1)AMAG=8.68589*ALOG(AMAG)
      IF(IPLT.EQ.0)GO TO 2100
      EFREQ=ALOG(FREQ)*0.434294
C
      IF(IPRT.GE.6) GO TO 2200
      IF(IPRT.NE.2) GO TO 2000
      AMAG=PHASE
2000  CONTINUE
C      CALL DRAW(EFREQ,AMAG,IPEN,4)
      IPEN=1
      GO TO 2200
2100  WRITE(IUNIT,2101) FREQ,AMAG,PHASE,U1,U2
2101  FORMAT(1X,G12.4,F12.5,F11.4,2G12.4)
2200  IFREQ=IFREQ+1
      IF(IFREQ.GT.NDEC)GO TO 2300
      PWR=IFREQ
      FTEM=10.0**(PWR/FLOAT(NDEC))
      FREQ=FMULT*FTEM
      IF(FREQ.GT.FSTOP) GO TO 2400
      GO TO 920
2300  FMULT=10.*FMULT
      IFREQ=0
      GO TO 2200
2400  RELM(INPT+3)=VIN
      RETURN
      END
      SUBROUTINE BJTAC
C .... LOAD AC BJT MODEL INTO Y & C ARRAYS
      INTEGER QI
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
```

APPENDIX E

```
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE (IELN(4),Q1)
C
      KPOS=IFRST(4)
      DO 2000 I=1,Q1
      MPOS=KPOS/2
      ITT=IELM(KPOS+4)
      KK=IELM(KPOS+7)
      LL=IELM(KPOS+8)
      MM=IELM(KPOS+5)
      CEEQ=0.0
      CCEQ=0.0
      BF=RELM(ITT+4)
      GMF=RELM(MPOS+7)
      GPIF=GMF/BF
      BR=RELM(ITT+5)
      GMR=RELM(MPOS+8)
      GPIR=GMR/BR
      IF(KK.LE.NNODE) GO TO 1100
      CEEQ=CEEQ-GMR*V(KK)
      KK=0
1100  IF(LL.LE.NNODE) GO TO 1200
      CCEQ=CCEQ+GMF*V(LL)
      CEEQ=CEEQ+GMR*V(LL)
      LL=0
1200  IF(MM.LE.NNODE) GO TO 1300
      CCEQ=CCEQ-GMF*V(MM)
      MM=0
1300  IF(KK.EQ.0)GO TO 1500
      Y(KK)=Y(KK)+GMR+GPIR
      C(KK)=C(KK)+(1.0+1.0/BR)*CEEQ-CCEQ
      IF(LL.EQ.0)GO TO 1400
      JJ= INDX(KK,LL)
      Y(JJ)=Y(JJ)+GMF-GMR-GPIR
      Y(NN)=Y(NN)-GPIR
1400  IF(MM.EQ.0)GO TO 1600
      JJ= INDX(KK,MM)
      Y(JJ)=Y(JJ)-GMF
      Y(NN)=Y(NN)-GMR
1500  IF(MM.EQ.0)GO TO 1600
      Y(MM)=Y(MM)+GMF+GPIF
      C(MM)=C(MM)+(1.0+1.0/BF)*CCEQ-CEEQ
      IF(LL.EQ.0)GO TO 1700
      JJ= INDX(MM,LL)
      Y(JJ)=Y(JJ)-GMF-GPIF+GMR
      Y(NN)=Y(NN)-GPIF
1600  IF(LL.EQ.0)GO TO 1700
      Y(LL)=Y(LL)+GPIF+GPIR
      C(LL)=C(LL)-CCEQ/BF-CEEQ/BR
1700  KPOS=IELM(KPOS+1)
2000  CONTINUE
      RETURN
      END
      SUBROUTINE DECAC
C .... PERFORMS LU DECOMPOSITION BASED ON RECORDED SPARSITY
C
      DOUBLE PRECISION DR,DI,UR,UI,AR
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
```

```
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1))
C
C     IUR  UPPER TRIANGULAR ROW ELEMENT COUNTER
C     IUC  UPPER TRIANGULAR ELEMENT COLUMN INDICATOR
C     ILC  LOWER TRIANGULAR COLUMN ELEMENT COUNTER
C     ILR  LOWER TRIANGULAR ELEMENT ROW INDICATOR
C
      IF(NNODE.EQ.1) GO TO 40
      NN=NNODE-1
      KNT=0
      DO 30 I=1,NN
      L=IORDR(I)
      UR=Y(L)
      UI=YI(L)
      DS=UR*UR+UI*UI
      IUS=IUR(I)+MLOC
      IUE=IUR(I+1)+MLOC
      IL=ILC(I)+NODE
      ILE=ILC(I+1)+NODE
C
C .... DOWN LOWER TRIANGLE COLUMNS
5     IF(IL.GE.ILE) GO TO 30
      DR=(Y(IL)*UR+YI(IL)*UI)/DS
      DI=(YI(IL)*UR-Y(IL)*UI)/DS
      Y(IL)=DR
      YI(IL)=DI
      IL=IL+1
      IU=IUS
C
C ....  ACROSS UPPER TRIANGLE ROWS
20    IF(IU.GE.IUE) GO TO 5
      KNT=KNT+1
      K=IPOS(KNT)
      AR=Y(K)-(Y(IU)*DR-YI(IU)*DI)
      YI(K)=YI(K)-(Y(IU)*DI+YI(IU)*DR)
      Y(K)=AR
      IU=IU+1
      GO TO 20
30    CONTINUE
40    RETURN
      END
      SUBROUTINE SOLAC
C
C .... PERFORMS FORWARD AND BACKWARD SUBSTITUTION
C      USING SPARSE POINTERS
C          MODIFIED FROM BIAS-N 5-19-77
      INTEGER*2 IELM
      DOUBLE PRECISION DR,DA,DB
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      DIMENSION IORDR(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
      EQUIVALENCE(NI(1,1),IORDR(1))
C
C .... FORWARD SUBSTITUTION
      NN=NNODE-1
      IF(NN.GT.0) GO TO 10
      DA=Y(1)
      DB=YI(1)
      DS=DA*DA+DB*DB
      DR=(C(1)*DA+CI(1)*DB)/DS
      VI(1)=(CI(1)*DA-C(1)*DB)/DS
      V(1)=DR
      GO TO 70
10    DO 30 I=1,NN
```

APPENDIX E

```
      L=IORDR(I)
      IL=ILC(I)
      ILE=ILC(I+1)
20    NL=IL+NODE
      IF(IL.GE.ILE) GO TO 30
      NR=ILR(IL)
      DR=C(NR)-(Y(NL)*C(L)-YI(NL)*CI(L))
      CI(NR)=CI(NR)-(Y(NL)*CI(L)+YI(NL)*C(L))
      C(NR)=DR
      IL=IL+I
      GO TO 20
30    CONTINUE
C
C .... BACK SUBSTUTION
      L=IORDR(NNODE)
      DA=Y(L)
      DB=YI(L)
      DS=DA*DA+DB*DB
      DR=(C(L)*DA+CI(L)*DB)/DS
      CI(L)=(CI(L)*DA-C(L)*DB)/DS
      C(L)=DR
      DO 50 I=1,NN
      NUI=NNODE-I
      L=IORDR(NUI)
      IU=IUR(NUI)
      IUE=IUR(NUI+I)
35    NL=IU+MLOC
      IF(IU.GE.IUE) GO TO 45
40    IC=IUC(IU)
      DR=C(L)-(Y(NL)*C(IC)-YI(NL)*CI(IC))
      CI(L)=CI(L)-(Y(NL)*CI(IC)+YI(NL)*C(IC))
      C(L)=DR
      IU=IU+1
      GO TO 35
45.   DA=Y(L)
      DB=YI(L)
      DS=DA*DA+DB*DB
      DR=(C(L)*DA+CI(L)*DB)/DS
      CI(L)=(CI(L)*DA-C(L)*DB)/DS
      C(L)=DR
50    CONTINUE
C .... TRANSFER INTO COMPLEX VOLTAGE VECTOR
C      DO 60 I=1,NNODE
C      V(I)=C(I)
C      VI(I)=CI(I)
C60    CONTINUE
70    RETURN
      END
      SUBROUTINE ADCPR
C .... ADD IMAGINARY CONDUCTANCE TO Y MATRIX
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     & (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      IF(KK.GT.NNODE)KK=0
      IF(LL.GT.NNODE)LL=0
      IF(KK.EQ.0)GOTO 100
      YI(KK)=YI(KK)+DS
      IF(LL.EQ.0)GOTO 200
      JJ= INDX(KK,LL)
      YI(JJ)=YI(JJ)-DS
      YI(NN)=YI(NN)-DS
100   IF(LL.EQ.0)GOTO 200
      YI(LL)=YI(LL)+DS
200   RETURN
      END
      SUBROUTINE ADCPC
```

```
C .... ADD IMAGINARY CURRENTS TO CURRENT VECTOR
      INTEGER*2 IELM
      DOUBLE PRECISION V(60),U(30),C(60),Y(600)
      DOUBLE PRECISION DELTA,DELT,DS,VI(1),CI(1),YI(1)
      DIMENSION ILC(1),ILR(1),RELM(1)
      COMMON U,C,Y,DS,DELT,DELTA
      COMMON T0,TEMP,DTEMP
      COMMON TM(6),A(8),CSAT,VT,VCT,TYPE
      COMMON IEL,JJ,KK,LL,MM,NN,IFLG,JFLG,ITT,ITER,IW,IR,IDISC,IUNIT
      COMMON IPLT,IPEN,LTYPE,ITEMP,ITOTL,IOUT,JOUT,INPT,IFORM
      COMMON MXLST,MXPOS,MXLOC,NDMAX,NODE,NNODE,IERR,MLOC,KPOS,LPOS
      COMMON IAQ(80),NI(30,2),IELN(9),IFRST(9),ILAST(9),IELM(1000)
      COMMON IUR(30),IUC(120),IPOS(400),NCON(30)
      EQUIVALENCE (IUR(1),ILC(1)),(IUC(1),ILR(1)),
     8 (IELM(1),RELM(1))
      EQUIVALENCE (C(1),V(1)),(C(30),CI(1),VI(1)),(Y(300),YI(1))
C
      IF(KK.GT.NNODE)KK=0
      IF(LL.GT.NNODE)LL=0
      IF(KK.EQ.0)GOTO 300
      CI(KK)=CI(KK)-DS
300   IF(LL.EQ.0)GOTO 400
      CI(LL)=CI(LL)+DS
400   RETURN
      END
```

DISTRIBUTION

ADMINISTRATOR
DEFENSE DOCUMENTATION CENTER
ATTN DDC-TCA (12 COPIES)
CAMERON STATION, BUILDING 5
ALEXANDRIA, VA 22314

COMMANDER
US ARMY RSCH & STD GP (EUR)
ATTN LTC JAMES M. KENNEDY, JR.
CHIEF, PHYSICS & MATH BRANCH
FPO NEW YORK 09510

COMMANDER
US ARMY MATERIEL DEVELOPMENT &
READINESS COMMAND
ATTN DRXAM-TL, HQ TECH LIBRARY
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

COMMANDER
US ARMY ARMAMENT MATERIEL
READINESS COMMAND
ATTN DRSAR-LEP-L, TECHNICAL LIBRARY
ROCK ISLAND, IL 61299

COMMANDER
US ARMY MISSILE & MUNITIONS
CENTER & SCHOOL
ATTN ATSK-CTD-F
REDSTONE ARSENAL, AL 35809

DIRECTOR
US ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY
ATTN DRXSY-MP
ABERDEEN PROVING GROUND, MD 21005

DIRECTOR
US ARMY BALLISTIC RESEARCH LABORATORY
ATTN DRDAR-TSB-S (STINFO)
ABERDEEN PROVING GROUND, MD 21005

TELEDYNE BROWN ENGINEERING
CUMMINGS RESEARCH PARK
ATTN DR. MELVIN L. PRICE, MS-44
HUNTSVILLE, AL 35807

COMMANDING OFFICER
NAVAL TRAINING EQUIPMENT CENTER
ATTN TECHNICAL LIBRARY
ORLANDO, FL 32813

US ARMY ELECTRONICS TECHNOLOGY
& DEVICES LABORATORY
ATTN DELET-DD
FORT MONMOUTH, NJ 07703

DIRECTOR
DEFENSE COMMUNICATIONS AGENCY
COMMAND & CONTROL TECHNICAL CENTER
ATTN TECHNICAL DIRECTOR
WASHINGTON, DC 20301

DIRECTOR
DEFENSE COMMUNICATIONS AGENCY
ATTN TECH LIBRARY
WASHINGTON, DC 20305

DIRECTOR
DEFENSE COMMUNICATIONS ENGINEERING CENTER
1860 WIEHLE AVE
ATTN RES & DEV
ATTN SYS CONT & GEN ENGR DIV
RESTON, VA 22090

DIRECTOR
NATIONAL SECURITY AGENCY
ATTN TECHNICAL LIBRARY
FORT GEORGE G. MEADE, MD 20755

OFFICE OF THE DEPUTY CHIEF OF STAFF
FOR RESEARCH, DEVELOPMENT & ACQ
DEPARTMENT OF THE ARMY
ATTN DAMA-ARZ-A, CHIEF SCIENTIST,
DA & DIRECTOR OF ARMY
RESEARCH, DR. M. E. LASSER
WASHINGTON, DC 20310

COMMANDER
ATMOSPHERIC SCIENCES LABORATORY
WHITE SANDS MISSILE RANGE, NM 88002

COMMANDER
US ARMY COMPUTER SYS COMMAND
MELPAR BUILDING
ATTN TECH LIB
FORT BELVOIR, VA 22060

DIRECTOR
ELECTRONIC WARFARE LABORATORY
FORT MONMOUTH, NJ 07703

COMMANDER
US ARMY MISSILE RESEARCH
& DEVELOPMENT COMMAND
ATTN DRDMI-T, DIR TECHNOLOGY LAB
ATTN DRDMI-E, DIR, ENGINEERING LAB
REDSTONE ARSENAL, AL 35809

DIRECTOR
NIGHT VISION AND ELECTRO-OPTICS LABORATORY
ATTN TECHNICAL LIBRARY
FORT BELVOIR, VA 22060

DISTRIBUTION (Cont'd)

COMMANDER
US ARMY ELECTRONICS PROVING GROUND
ATTN STEEP-MT-A, METHOD & INSTR BR
FORT HUACHUCA, AZ 85613

PROFESSOR H. C. LIN (2 COPIES)
ELECTRICAL ENGINEERING DEPT
UNIVERSITY OF MARYLAND
COLLEGE PARK, MD 20792

PROFESSOR D. O. PEDERSON (2 COPIES)
EECS DEPT, CORY HALL
UNIVERSITY OF CALIFORNIA
BERKELEY, CA 94720

PROFESSOR R. W. DUTTON (2 COPIES)
INTEGRATED CIRCUITS LABORATORY
STANFORD UNIVERSITY
STANFORD, CA 94305

US ARMY ELECTRONICS RESEARCH
& DEVELOPMENT COMMAND
ATTN WISEMAN, ROBERT S., DR., DRDEL-CT

HARRY DIAMOND LABORATORIES
ATTN 00100, COMMANDER/TECH DIR/TSO
ATTN CHIEF, DIV 10000
ATTN CHIEF, DIV 20000
ATTN CHIEF, DIV 30000
ATTN CHIEF, DIV 40000
ATTN RECORD COPY, 81200
ATTN HDL LIBRARY, (3 COPIES) 81100
ATTN HDL LIBRARY, (WOODBRIDGE) 81100
ATTN TECHNICAL REPORTS BRANCH, 81300
ATTN CHAIRMAN, EDITORIAL COMMITTEE
ATTN CHIEF, 13000 (2 COPIES)
ATTN CHIEF, 13500
ATTN DOBRIANSKY, 13500
ATTN COOK, D. R., 11100
ATTN CAIRNS, C. W., 11100
ATTN ROSEN, R., 48100
ATTN CHOY, S., 48100
ATTN FURLANI, J., 48100
ATTN PEPERONE, S. J., 36000
ATTN DENT, J., 36100
ATTN MCNALLY, G., 36200
ATTN GOODMAN, R., 34400
ATTN INGERSOLL, P, 34300
ATTN SCOTT, W. J., 21500
ATTN DANDO, J., 21400
ATTN WICKLUND, J., 22100
ATTN HALPIN, J., 22800 (5 COPIES)
ATTN HANSEN, R., 15300
ATTN SHREVE, J., 15300
ATTN HITE, J., 15300
ATTN SANN, K., 11100 (2 COPIES)
ATTN GOTO, J., 13400
ATTN HOFF, R. S., 47000
ATTN COX, L. S., 00210
ATTN BIEHL, B. L., 15300 (20 COPIES)